In [21]:
```
#loaddata
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPool2D,
Dropout
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
classes_name = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
 'frog', 'horse', 'ship', 'truck']
X_train.max()
X_train = X_train/255
X_test = X_test/255
y_test
```

Out[21]:
```
array([[3],
       [8],
       [8],
       ...,
       [5],
       [1],
       [7]], dtype=uint8)
```

In [22]:
```
#CNN MODEL
#Input
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3, 3), padding='same', activa
tion='relu', input_shape = [32, 32, 3]))
model.add(MaxPool2D(pool_size=(2,2), strides=2, padding='valid'))
#layer2
model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activa
tion='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=2, padding='valid'))
#layer3
```

```
model.add(Conv2D(filters=64, kernel_size=(3, 3), padding='same', activa
tion='relu'))


#layer4
model.add(Flatten())
model.add(Dense(units = 64, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 8, 64)          36928
_________________________________________________________________
flatten_4 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                262208
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
Total params: 319,178
Trainable params: 319,178
Non-trainable params: 0
_________________________________________________________________
```

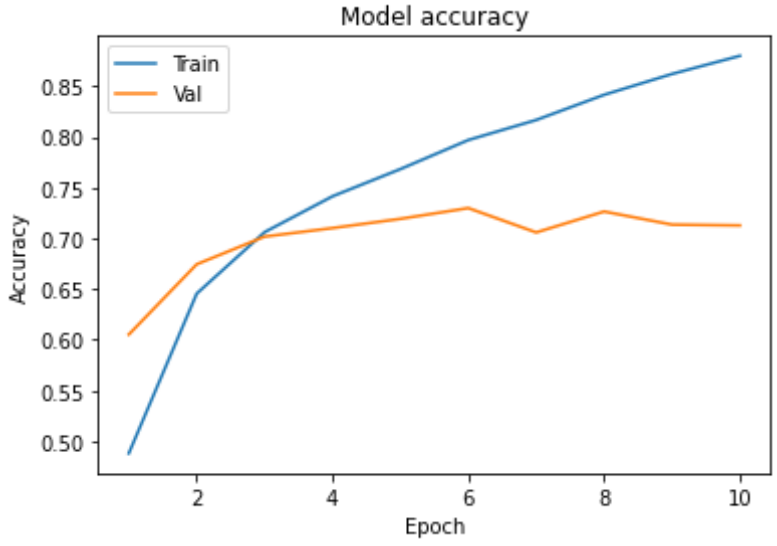
In [23]:
```
#compile model
model.compile(optimizer='adam', loss = 'sparse_categorical_crossentrop
y', metrics=['sparse_categorical_accuracy'])
```

In [24]:
```
#model fitting
history = model.fit(X_train, y_train, batch_size=32, epochs=10, verbose
=1, validation_data=(X_test, y_test))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 122s 2ms/sample - loss:
1.4109 - sparse_categorical_accuracy: 0.4881 - val_loss: 1.1083 - val_s
parse_categorical_accuracy: 0.6052
Epoch 2/10
50000/50000 [==============================] - 132s 3ms/sample - loss:
1.0069 - sparse_categorical_accuracy: 0.6455 - val_loss: 0.9169 - val_s
parse_categorical_accuracy: 0.6744
Epoch 3/10
50000/50000 [==============================] - 132s 3ms/sample - loss:
0.8423 - sparse_categorical_accuracy: 0.7061 - val_loss: 0.8687 - val_s
parse_categorical_accuracy: 0.7017
Epoch 4/10
50000/50000 [==============================] - 130s 3ms/sample - loss:
0.7398 - sparse_categorical_accuracy: 0.7414 - val_loss: 0.8419 - val_s
parse_categorical_accuracy: 0.7101
Epoch 5/10
50000/50000 [==============================] - 130s 3ms/sample - loss:
0.6566 - sparse_categorical_accuracy: 0.7682 - val_loss: 0.8166 - val_s
parse_categorical_accuracy: 0.7192
Epoch 6/10
50000/50000 [==============================] - 129s 3ms/sample - loss:
0.5785 - sparse_categorical_accuracy: 0.7968 - val_loss: 0.8473 - val_s
parse_categorical_accuracy: 0.7298
Epoch 7/10
50000/50000 [==============================] - 130s 3ms/sample - loss:
0.5185 - sparse_categorical_accuracy: 0.8166 - val_loss: 0.9110 - val_s
parse_categorical_accuracy: 0.7058
Epoch 8/10
50000/50000 [==============================] - 134s 3ms/sample - loss:
0.4495 - sparse_categorical_accuracy: 0.8414 - val_loss: 0.8822 - val_s
parse_categorical_accuracy: 0.7264
Epoch 9/10
50000/50000 [==============================] - 131s 3ms/sample - loss:
```

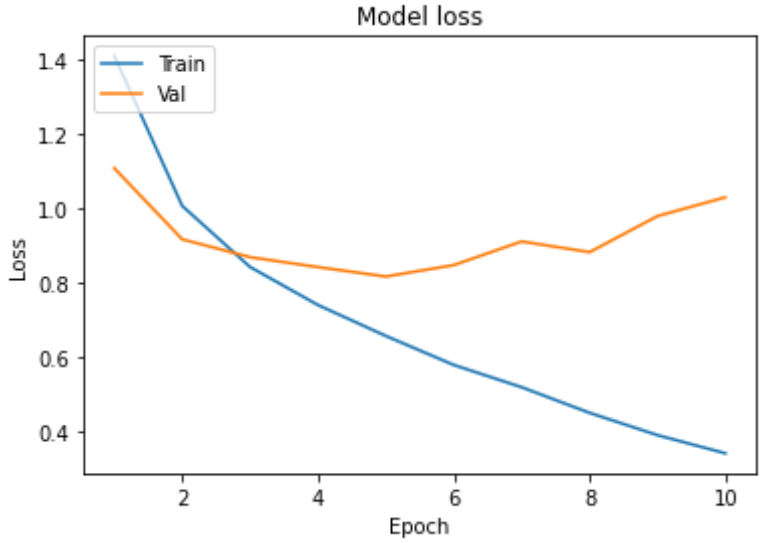
```
0.3895 - sparse_categorical_accuracy: 0.8620 - val_loss: 0.9793 - val_s
parse_categorical_accuracy: 0.7135
Epoch 10/10
50000/50000 [==============================] - 133s 3ms/sample - loss:
0.3405 - sparse_categorical_accuracy: 0.8798 - val_loss: 1.0299 - val_s
parse_categorical_accuracy: 0.7127
```

In [27]:
```
# Plot training & validation accuracy values
epoch_range = range(1, 11)
plt.plot(epoch_range, history.history['sparse_categorical_accuracy'])
plt.plot(epoch_range, history.history['val_sparse_categorical_accuracy'
])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



In [28]:
```
# Plot training & validation loss values
plt.plot(epoch_range, history.history['loss'])
plt.plot(epoch_range, history.history['val_loss'])
plt.title('Model loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



In [29]:
```
#confusionmatrix
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
y_pred = model.predict_classes(X_test)
y_test
mat = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(mat,figsize=(9,9), class_names=classes_name, show
_normed=True)
```

Out[29]:
```
(<Figure size 648x648 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f5efe064828>)
```