In [1]:
```
import numpy as np
import tensorflow
from matplotlib import pyplot as plt
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Batc
hNormalization
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.regularizers import l2
from tensorflow.keras.datasets import cifar100
```

In [2]:
```
(trainX, trainY), (testX, testY) = cifar100.load_data()
```

In [3]:
```
# Target classes: numbers to text
# Source: https://github.com/keras-team/keras/issues/2653#issuecomment-4501339
96
classes = [
  'apple',
  'aquarium_fish',
  'baby',
  'bear',
  'beaver',
  'bed',
  'bee',
  'beetle',
  'bicycle',
  'bottle',
  'bowl',
  'boy',
  'bridge',
  'bus',
  'butterfly',
  'camel',
  'can',
  'castle',
  'caterpillar',
  'cattle',
  'chair',
  'chimpanzee',
  'clock',
  'cloud',
  'cockroach',
  'couch',
  'crab',
  'crocodile',
  'cup',
  'dinosaur',
  'dolphin',
  'elephant',
  'flatfish',
  'forest',
  'fox',
  'girl',
  'hamster',
  'house',
  'kangaroo',
  'computer_keyboard',
  'lamp',
  'lawn_mower',
  'leopard',
  'lion',
  'lizard',
  'lobster',
  'man',
  'maple_tree',
  'motorcycle',
  'mountain',
  'mouse',
  'mushroom',
```

```
  'oak_tree',
  'orange',
  'orchid',
  'otter',
  'palm_tree',
  'pear',
  'pickup_truck',
  'pine_tree',
  'plain',
  'plate',
  'poppy',
  'porcupine',
  'possum',
  'rabbit',
  'raccoon',
  'ray',
  'road',
  'rocket',
  'rose',
  'sea',
  'seal',
  'shark',
  'shrew',
  'skunk',
  'skyscraper',
  'snail',
  'snake',
  'spider',
  'squirrel',
  'streetcar',
  'sunflower',
  'sweet_pepper',
  'table',
  'tank',
  'telephone',
  'television',
  'tiger',
  'tractor',
  'train',
  'trout',
  'tulip',
  'turtle',
  'wardrobe',
  'whale',
  'willow_tree',
  'wolf',
  'woman',
  'worm',
]
```

In [4]:
```
# Visualize firt 9 images
for i in range(9):
    plt.subplot(330 + 1 + i)
    image = trainX[i]
    target = trainY[i][0]
    plt.axis('off')
    plt.imshow(image)
    plt.title(f'{classes[target]}')
plt.show()
```

In [5]:
```
trainX = trainX.reshape(trainX.shape[0], 32,32,3)
trainY = trainY.reshape(trainY.shape[0], 1)
testX = testX.reshape(testY.shape[0], 32, 32, 3)
testY = testY.reshape(testY.shape[0], 1)
```

In [6]:
```
# Parse numbers as floats
trainX = trainX.astype('float32')
testX = testX.astype('float32')

# Normalize data
trainX = trainX / 255.0
testX = testX / 255.0
```

In [7]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape = (32, 32, 3), kernel_regularizer = l
2(0.005), activation = 'elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Conv2D(128, (3, 3), kernel_regularizer = l2(0.005), activation = 'el
u'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Conv2D(128, (3, 3), kernel_regularizer = l2(0.005), activation = 'el
u'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), kernel_regularizer = l2(0.005), activation = 'el
u'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Flatten())
model.add(Dense(units = 128, activation = 'elu'))
model.add(Dense(units = 256, activation = 'elu'))
model.add(Dense(units = 100, activation = 'softmax'))
```

In [8]:
```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', me
trics = ['accuracy'])
```

In [9]:
```
rlrop = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.1, patience = 10)
```

In [10]:
```
early_stop = EarlyStopping(monitor = 'val_loss',
                           mode = 'min',
                           verbose = 1,
                           patience = 30)
```

In [11]:
```
history = model.fit(trainX, trainY,
                         steps_per_epoch = 400,
                         epochs = 100,
                         #verbosity = 1,
                         validation_split = 0.2,
                         callbacks = [rlrop, early_stop])
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/100
40000/40000 [==============================] - 126s 3ms/sample - loss: 4.3252
- accuracy: 0.1828 - val_loss: 5.2625 - val_accuracy: 0.0700
Epoch 2/100
40000/40000 [==============================] - 104s 3ms/sample - loss: 3.2391
- accuracy: 0.2998 - val_loss: 4.1788 - val_accuracy: 0.1761
Epoch 3/100
40000/40000 [==============================] - 104s 3ms/sample - loss: 3.0300
- accuracy: 0.3465 - val_loss: 4.3409 - val_accuracy: 0.1775
Epoch 4/100
40000/40000 [==============================] - 106s 3ms/sample - loss: 2.9066
- accuracy: 0.3832 - val_loss: 3.3714 - val_accuracy: 0.2982
Epoch 5/100
40000/40000 [==============================] - 107s 3ms/sample - loss: 2.8178
- accuracy: 0.4107 - val_loss: 3.2840 - val_accuracy: 0.3373
Epoch 6/100
40000/40000 [==============================] - 107s 3ms/sample - loss: 2.7530
- accuracy: 0.4304 - val_loss: 3.5828 - val_accuracy: 0.2937
Epoch 7/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 2.6902
- accuracy: 0.4507 - val_loss: 3.6019 - val_accuracy: 0.2947
Epoch 8/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 2.6293
- accuracy: 0.4685 - val_loss: 3.5475 - val_accuracy: 0.3119
Epoch 9/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 2.5816
- accuracy: 0.4820 - val_loss: 3.2323 - val_accuracy: 0.3699
Epoch 10/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 2.5296
- accuracy: 0.4986 - val_loss: 3.4871 - val_accuracy: 0.3417
Epoch 11/100
40000/40000 [==============================] - 116s 3ms/sample - loss: 2.4882
- accuracy: 0.5110 - val_loss: 3.4263 - val_accuracy: 0.3437
Epoch 12/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 2.4410
- accuracy: 0.5217 - val_loss: 3.5240 - val_accuracy: 0.3458
Epoch 13/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 2.3977
- accuracy: 0.5376 - val_loss: 3.7135 - val_accuracy: 0.3152
Epoch 14/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 2.3695
- accuracy: 0.5473 - val_loss: 3.5284 - val_accuracy: 0.3522
Epoch 15/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 2.3311
- accuracy: 0.5572 - val_loss: 4.1948 - val_accuracy: 0.2615
Epoch 16/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 2.2837
- accuracy: 0.5714 - val_loss: 3.6979 - val_accuracy: 0.3291
Epoch 17/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 2.2457
- accuracy: 0.5809 - val_loss: 3.6579 - val_accuracy: 0.3471
Epoch 18/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 2.2117
- accuracy: 0.5953 - val_loss: 3.4908 - val_accuracy: 0.3743
Epoch 19/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 2.1848
```

```
- accuracy: 0.6023 - val_loss: 3.8797 - val_accuracy: 0.3349
Epoch 20/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 1.6245
- accuracy: 0.7621 - val_loss: 2.8346 - val_accuracy: 0.4920
Epoch 21/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 1.3456
- accuracy: 0.8293 - val_loss: 2.7888 - val_accuracy: 0.4940
Epoch 22/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 1.1831
- accuracy: 0.8631 - val_loss: 2.7597 - val_accuracy: 0.4916
Epoch 23/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 1.0601
- accuracy: 0.8828 - val_loss: 2.7660 - val_accuracy: 0.4902
Epoch 24/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.9537
- accuracy: 0.9054 - val_loss: 2.7583 - val_accuracy: 0.4861
Epoch 25/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.8592
- accuracy: 0.9222 - val_loss: 2.8006 - val_accuracy: 0.4859
Epoch 26/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.7780
- accuracy: 0.9370 - val_loss: 2.8942 - val_accuracy: 0.4713
Epoch 27/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.7146
- accuracy: 0.9471 - val_loss: 2.9108 - val_accuracy: 0.4803
Epoch 28/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.6554
- accuracy: 0.9561 - val_loss: 2.9620 - val_accuracy: 0.4747
Epoch 29/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.6076
- accuracy: 0.9629 - val_loss: 2.9858 - val_accuracy: 0.4788
Epoch 30/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.5648
- accuracy: 0.9691 - val_loss: 3.0552 - val_accuracy: 0.4682
Epoch 31/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.5256
- accuracy: 0.9735 - val_loss: 3.0819 - val_accuracy: 0.4673
Epoch 32/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.4995
- accuracy: 0.9735 - val_loss: 3.1257 - val_accuracy: 0.4728
Epoch 33/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.4754
- accuracy: 0.9768 - val_loss: 3.1471 - val_accuracy: 0.4680
Epoch 34/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.4442
- accuracy: 0.9816 - val_loss: 3.2416 - val_accuracy: 0.4683
Epoch 35/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.3966
- accuracy: 0.9916 - val_loss: 3.1433 - val_accuracy: 0.4812
Epoch 36/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.3803
- accuracy: 0.9952 - val_loss: 3.1494 - val_accuracy: 0.4813
Epoch 37/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.3728
- accuracy: 0.9965 - val_loss: 3.1493 - val_accuracy: 0.4798
Epoch 38/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3674
```

```
- accuracy: 0.9967 - val_loss: 3.1642 - val_accuracy: 0.4828
Epoch 39/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3614
- accuracy: 0.9972 - val_loss: 3.1652 - val_accuracy: 0.4828
Epoch 40/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3565
- accuracy: 0.9970 - val_loss: 3.1896 - val_accuracy: 0.4803
Epoch 41/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3501
- accuracy: 0.9978 - val_loss: 3.1823 - val_accuracy: 0.4812
Epoch 42/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3453
- accuracy: 0.9975 - val_loss: 3.1954 - val_accuracy: 0.4819
Epoch 43/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.3405
- accuracy: 0.9979 - val_loss: 3.1892 - val_accuracy: 0.4806
Epoch 44/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.3357
- accuracy: 0.9980 - val_loss: 3.2206 - val_accuracy: 0.4770
Epoch 45/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3297
- accuracy: 0.9985 - val_loss: 3.2055 - val_accuracy: 0.4814
Epoch 46/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.3293
- accuracy: 0.9988 - val_loss: 3.2074 - val_accuracy: 0.4809
Epoch 47/100
40000/40000 [==============================] - 112s 3ms/sample - loss: 0.3288
- accuracy: 0.9987 - val_loss: 3.2057 - val_accuracy: 0.4813
Epoch 48/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.3279
- accuracy: 0.9987 - val_loss: 3.2077 - val_accuracy: 0.4806
Epoch 49/100
40000/40000 [==============================] - 110s 3ms/sample - loss: 0.3263
- accuracy: 0.9987 - val_loss: 3.2090 - val_accuracy: 0.4803
Epoch 50/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3267
- accuracy: 0.9987 - val_loss: 3.2099 - val_accuracy: 0.4807
Epoch 51/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3255
- accuracy: 0.9987 - val_loss: 3.2109 - val_accuracy: 0.4812
Epoch 52/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.3251
- accuracy: 0.9985 - val_loss: 3.2109 - val_accuracy: 0.4821
Epoch 53/100
40000/40000 [==============================] - 109s 3ms/sample - loss: 0.3247
- accuracy: 0.9986 - val_loss: 3.2120 - val_accuracy: 0.4811
Epoch 54/100
40000/40000 [==============================] - 108s 3ms/sample - loss: 0.3242
- accuracy: 0.9986 - val_loss: 3.2163 - val_accuracy: 0.4820
Epoch 00054: early stopping
```

In [12]:

```
model.save('Version_Final.h5')
```

In [13]:
```
model.summary()
```

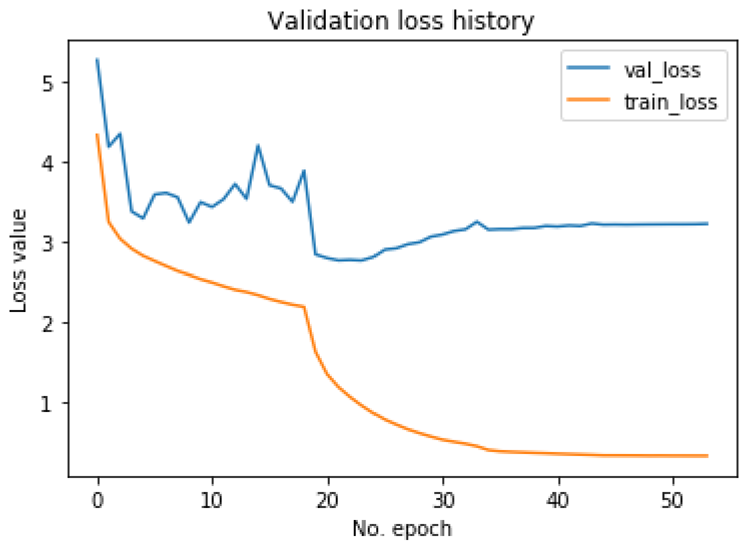
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896       
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 128)       36992     
_________________________________________________________________
batch_normalization_1 (Batch (None, 13, 13, 128)       512       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 128)         0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         147584    
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 128)         512       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 256)         295168    
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 256)         1024      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 256)               0         
_________________________________________________________________
dense (Dense)                (None, 128)               32896     
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024     
_________________________________________________________________
dense_2 (Dense)              (None, 100)               25700     
=================================================================
Total params: 574,436
Trainable params: 573,348
Non-trainable params: 1,088
_________________________________________________________________
```

In [14]:
```
# Generate generalization metrics
score = model.evaluate(testX, testY, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

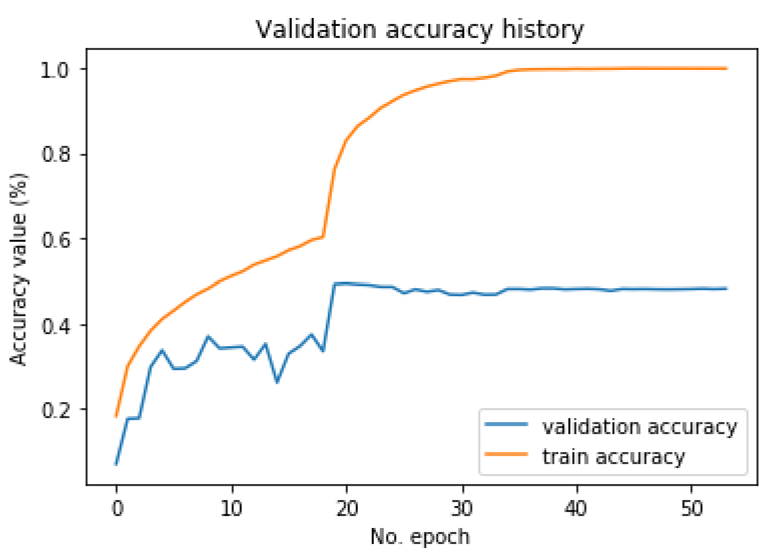
```
Test loss: 3.152114510726929 / Test accuracy: 0.4839000105857849
```

In [15]:
```
testY_pred = model.predict(testX)
```

In [16]:
```
# Visualize history
# Plot history: Loss
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.plot(history.history['loss'], label = 'train_loss')
plt.title('Validation loss history')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend()
plt.show()
```



In [17]:
```
# Plot history: Accuracy
plt.plot(history.history['val_accuracy'], label = 'validation accuracy')
plt.plot(history.history['accuracy'], label = 'train accuracy')
plt.title('Validation accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.legend()
plt.show()
```



In [18]:
```
import random
fig=plt.figure(figsize=(10,7))
columns = 3
rows = 3
# Visualize firt 9 images
for i in range(1,10):
    fig.add_subplot(rows, columns, i)
    k = random.randrange(10000)
    image = testX[k]
    target_pred = testY_pred[k].tolist().index(testY_pred[k].max())
    target_actual = testY[k][0]
    plt.axis('off')
    plt.imshow(image)
    plt.title(classes[target_pred]+' ('+classes[target_actual]+')')
plt.show()
```