# Introduction:

1. In this assignment, you will build Convolutional Neural Network to classify CIFAR-10 Images.
2. You can directly load dataset from many deep learning packages.
3. You can use any deep learning packages such as pytorch, keras or tensorflow for this assignment.

# Requirements:

1. You need to load cifar 10 data and split the entire training dataset into training and validation.
2. You will implement a CNN model to classify cifar 10 images with provided structure.
3. You need to plot the training and validation accuracy or loss obtained from above step.
4. Then you can use tuned hyper-parameters to train using the entire training dataset.
5. You should report the testing accuracy using the model with complete data.
6. You may try to change the structure (e.g, add BN layer or dropout layer,...) and analyze your findings.

# Google Colab

- If you do not have GPU, the training of a CNN can be slow. Google Colab is a good option.

# Batch Normalization (BN)

## Background:

- Batch Normalization is a technique to speed up training and help make the model more stable.
- In simple words, batch normalization is just another network layer that gets inserted between a hidden layer and the next hidden layer. Its job is to take the outputs from the first hidden layer and normalize them before passing them on as the input of the next hidden layer.
- For more detailed information, you may refer to the original paper: https://arxiv.org/pdf/1502.03167.pdf (https://arxiv.org/pdf/1502.03167.pdf).

## BN Algorithm:

- Input: Values of $x$ over a mini-batch: $\mathbf{B} = \{x_1, \ldots, x_m\}$;
- Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$, $\gamma, \beta$ are learnable parameters

Normalization of the Input:

$$\mu_{\mathbf{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathbf{B}}^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathbf{B}})^2$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathbf{B}}}{\sqrt{\sigma_{\mathbf{B}}}^2 + \epsilon}$$

Re-scaling and Offsetting:

$$y_i = \gamma \hat{x}_i + \beta = BN_{\gamma,\beta}(x_i)$$

## Advantages of BN:

1. Improves gradient flow through the network.
2. Allows use of saturating nonlinearities and higher learning rates.
3. Makes weights easier to initialize.
4. Act as a form of regularization and may reduce the need for dropout.

## Implementation:

- The batch normalization layer has already been implemented in many packages. You may simply call the function to build the layer. For example: torch.nn.BatchNorm2d() using pytroch package, keras.layers.BatchNormalization() using keras package.
- The location of BN layer: Please make sure `BatchNormalization` is between a `Conv` / `Dense` layer and an `activation` layer.

# 1. Data preparation

## 1.1. Load data

In [3]:
```
# Load Cifar-10 Data
# This is just an example, you may load dataset from other packages.
import keras
import numpy as np
import tensorflow.keras

### If you can not load keras dataset, un-comment these two lines.
#import ssl
#ssl._create_default_https_context = ssl._create_unverified_context

(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

print('shape of x_train: ' + str(x_train.shape))
print('shape of y_train: ' + str(y_train.shape))
print('shape of x_test: ' + str(x_test.shape))
print('shape of y_test: ' + str(y_test.shape))
print('number of classes: ' + str(np.max(y_train) - np.min(y_train) + 1))
```

```
shape of x_train: (50000, 32, 32, 3)
shape of y_train: (50000, 1)
shape of x_test: (10000, 32, 32, 3)
shape of y_test: (10000, 1)
number of classes: 10
```

## 1.2. One-hot encode the labels (5 points)

In the input, a label is a scalar in $\{0, 1, \cdots, 9\}$. One-hot encode transform such a scalar to a $10$-dim vector. E.g., a scalar `y_train[j]=3` is transformed to the vector `y_train_vec[j]=[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]` .

1. Implement a function `to_one_hot` that transforms an $n \times 1$ array to a $n \times 10$ matrix.
2. Apply the function to `y_train` and `y_test` .

In [4]:
```
def to_one_hot(y, num_class=10):
    y_new = []
    for val in y:
        tempArr = np.zeros(num_class)
        tempArr[val] = 1
        y_new.append(tempArr)

    return np.asarray(y_new)
    pass

x_train, x_test = x_train.astype('float32') / 255, x_test.astype('float32') /
y_train_vec = to_one_hot(y_train)
y_test_vec = to_one_hot(y_test)

print('Shape of y_train_vec: ' + str(y_train_vec.shape))
print('Shape of y_test_vec: ' + str(y_test_vec.shape))

print(y_train[0])
print(y_train_vec[0])
```

```
Shape of y_train_vec: (50000, 10)
Shape of y_test_vec: (10000, 10)
[6]
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

**Remark: the outputs should be**

- Shape of y_train_vec: (50000, 10)
- Shape of y_test_vec: (10000, 10)
- [6]
- [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

## 1.3. Randomly partition the training set to training and validation sets (5 points)

Randomly partition the 50K training samples to 2 sets:

- a training set containing 40K samples: x_tr, y_tr
- a validation set containing 10K samples: x_val, y_val

In [5]:
```
from sklearn.model_selection import train_test_split

x_tr, x_val, y_tr, y_val = train_test_split(x_train,y_train_vec,test_size=0.2,

print('Shape of x_tr: ' + str(x_tr.shape))
print('Shape of y_tr: ' + str(y_tr.shape))
print('Shape of x_val: ' + str(x_val.shape))
print('Shape of y_val: ' + str(y_val.shape))
```

```
Shape of x_tr: (40000, 32, 32, 3)
Shape of y_tr: (40000, 10)
Shape of x_val: (10000, 32, 32, 3)
Shape of y_val: (10000, 10)
```

# 2. Build a CNN and tune its hyper-parameters (50 points)

- Build a convolutional neural network model using the below structure:
- It should have a structure of: Conv - ReLU - Max Pool - ConV - ReLU - Max Pool - Dense - ReLU - Dense - Softmax
- In the graph 3@32x32 means the dimension of input image, 32@30x30 means it has 32 filters and the dimension now becomes 30x30 after the convolution.
- All convolutional layers (Conv) should have stride = 1 and no padding.
- Max Pooling has a pool size of 2 by 2.

- You may use the validation data to tune the hyper-parameters (e.g., learning rate, and optimization algorithm)
- Do NOT use test data for hyper-parameter tuning!!!
- Try to achieve a validation accuracy as high as possible.

In [6]:
```
# Build the model
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Activa
from keras.models import Sequential

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (4, 4), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(256, activation='relu'))

model.add(Dense(10, activation='softmax'))

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, 12, 12, 64)        32832     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 2304)              0         
                                                                 
 dense (Dense)               (None, 256)               590080    
                                                                 
 dense_1 (Dense)             (None, 10)                2570      
                                                                 
=================================================================
Total params: 626,378
Trainable params: 626,378
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```
# Define model optimizer and loss function
from tensorflow.keras import optimizers

lr = 0.0001
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(le
```

In [8]:
```
# Train the model and store model parameters/loss values
model_1 = model.fit(x_tr, y_tr, batch_size=128, epochs=50, validation_data=(x_
model.save('model_1.h5')
```

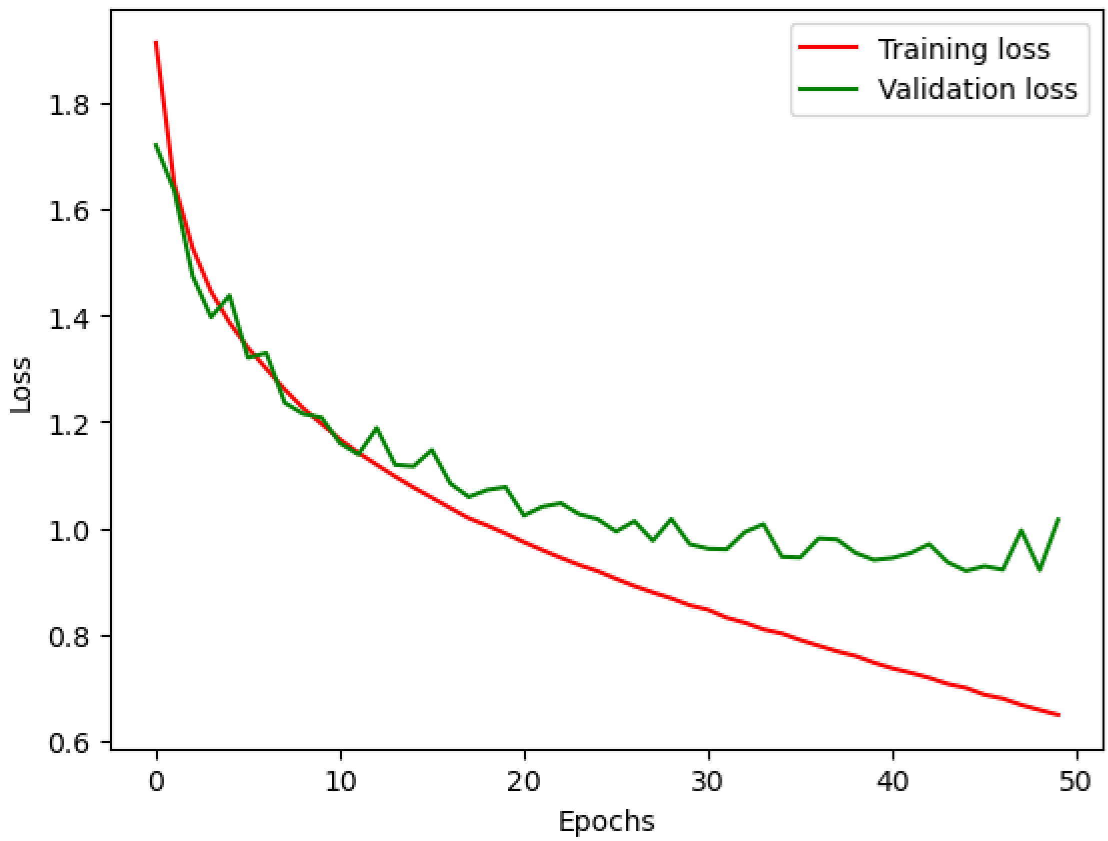
```
Epoch 1/50
313/313 [==============================] - 7s 6ms/step - loss: 1.9130 - ac
curacy: 0.3328 - val_loss: 1.7204 - val_accuracy: 0.3934
Epoch 2/50
313/313 [==============================] - 1s 4ms/step - loss: 1.6476 - ac
curacy: 0.4212 - val_loss: 1.6346 - val_accuracy: 0.4177
Epoch 3/50
313/313 [==============================] - 2s 5ms/step - loss: 1.5266 - ac
curacy: 0.4651 - val_loss: 1.4741 - val_accuracy: 0.4801
Epoch 4/50
313/313 [==============================] - 1s 4ms/step - loss: 1.4451 - ac
curacy: 0.4913 - val_loss: 1.3970 - val_accuracy: 0.5091
Epoch 5/50
313/313 [==============================] - 1s 4ms/step - loss: 1.3866 - ac
curacy: 0.5149 - val_loss: 1.4378 - val_accuracy: 0.4946
Epoch 6/50
313/313 [==============================] - 1s 4ms/step - loss: 1.3395 - ac
curacy: 0.5310 - val_loss: 1.3207 - val_accuracy: 0.5356
Epoch 7/50
```

## 3. Plot the training and validation loss curve versus epochs. (5 points)

In [9]:
```
model_1.history.keys()
```

Out[9]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [10]:
```
# Plot the loss curve

import matplotlib.pyplot as plt
#%matplotlib inline

loss = model_1.history['loss']
val_loss = model_1.history['val_loss']

plt.xlabel('Epochs')
plt.ylabel('Loss')

epochs = range(len(loss))

plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'green', label='Validation loss')

plt.legend()
plt.show()
```




## 4. Train (again) and evaluate the model (5 points)

- To this end, you have found the "best" hyper-parameters.
- Now, fix the hyper-parameters and train the network on the entire training set (all the 50K training samples)
- Evaluate your model on the test set.

## Train the model on the entire training set

Why? Previously, you used 40K samples for training; you wasted 10K samples for the sake of hyper-parameter tuning. Now you already know the hyper-parameters, so why not using all the 50K samples for training?

In [11]:
```
#<Compile your model again (using the same hyper-parameters you tuned above)>
model = Sequential()
model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape=(32, 32, 3)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (4, 4), activation = 'relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(256, activation = 'relu'))
model.add(Dense(10, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(le
```

In [12]:
```
#<Train your model on the entire training set (50K samples)>
model_2 = model.fit(x_train, y_train_vec, batch_size=128, epochs=50)
model.save('model_2.h5')
```

```
Epoch 1/50
391/391 [==============================] - 2s 4ms/step - loss: 1.8399 - acc:
0.3516
Epoch 2/50
391/391 [==============================] - 1s 4ms/step - loss: 1.5559 - acc:
0.4488
Epoch 3/50
391/391 [==============================] - 1s 4ms/step - loss: 1.4341 - acc:
0.4925
Epoch 4/50
391/391 [==============================] - 1s 4ms/step - loss: 1.3553 - acc:
0.5208
Epoch 5/50
391/391 [==============================] - 1s 4ms/step - loss: 1.2973 - acc:
0.5439
Epoch 6/50
391/391 [==============================] - 1s 4ms/step - loss: 1.2485 - acc:
0.5618
Epoch 7/50
391/391 [==============================] - 1s 4ms/step - loss: 1.2071 - acc:
0.5779
Epoch 8/50
391/391 [==============================] - 1s 4ms/step - loss: 1.1724 - acc:
0.5905
Epoch 9/50
391/391 [==============================] - 1s 4ms/step - loss: 1.1408 - acc:
0.6011
Epoch 10/50
391/391 [==============================] - 1s 4ms/step - loss: 1.1099 - acc:
0.6126
Epoch 11/50
391/391 [==============================] - 2s 4ms/step - loss: 1.0856 - acc:
0.6223
Epoch 12/50
391/391 [==============================] - 1s 4ms/step - loss: 1.0612 - acc:
0.6311
Epoch 13/50
391/391 [==============================] - 1s 4ms/step - loss: 1.0376 - acc:
0.6397
Epoch 14/50
391/391 [==============================] - 1s 4ms/step - loss: 1.0159 - acc:
0.6481
Epoch 15/50
391/391 [==============================] - 1s 4ms/step - loss: 0.9958 - acc:
0.6558
Epoch 16/50
391/391 [==============================] - 1s 4ms/step - loss: 0.9778 - acc:
0.6607
Epoch 17/50
391/391 [==============================] - 1s 4ms/step - loss: 0.9600 - acc:
0.6700
Epoch 18/50
391/391 [==============================] - 1s 4ms/step - loss: 0.9430 - acc:
0.6756
Epoch 19/50
391/391 [==============================] - 1s 4ms/step - loss: 0.9294 - acc:
0.6785
```

```
Epoch 20/50
391/391 [==============================] - 1s 4ms/step - loss: 0.9124 - acc:
0.6854
Epoch 21/50
391/391 [==============================] - 1s 4ms/step - loss: 0.8999 - acc:
0.6893
Epoch 22/50
391/391 [==============================] - 1s 4ms/step - loss: 0.8844 - acc:
0.6950
Epoch 23/50
391/391 [==============================] - 1s 4ms/step - loss: 0.8714 - acc:
0.7007
Epoch 24/50
391/391 [==============================] - 1s 4ms/step - loss: 0.8576 - acc:
0.7061
Epoch 25/50
391/391 [==============================] - 2s 4ms/step - loss: 0.8459 - acc:
0.7098
Epoch 26/50
391/391 [==============================] - 2s 4ms/step - loss: 0.8335 - acc:
0.7132
Epoch 27/50
391/391 [==============================] - 2s 4ms/step - loss: 0.8240 - acc:
0.7170
Epoch 28/50
391/391 [==============================] - 1s 4ms/step - loss: 0.8111 - acc:
0.7214
Epoch 29/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7995 - acc:
0.7250
Epoch 30/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7879 - acc:
0.7289
Epoch 31/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7783 - acc:
0.7339
Epoch 32/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7673 - acc:
0.7366
Epoch 33/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7559 - acc:
0.7401
Epoch 34/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7456 - acc:
0.7462
Epoch 35/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7358 - acc:
0.7494
Epoch 36/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7263 - acc:
0.7518
Epoch 37/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7146 - acc:
0.7542
Epoch 38/50
391/391 [==============================] - 1s 4ms/step - loss: 0.7052 - acc:
0.7596
```

```
Epoch 39/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6952 - acc:
0.7621
Epoch 40/50
391/391 [==============================] - 2s 4ms/step - loss: 0.6857 - acc:
0.7660
Epoch 41/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6753 - acc:
0.7676
Epoch 42/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6664 - acc:
0.7737
Epoch 43/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6557 - acc:
0.7757
Epoch 44/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6468 - acc:
0.7783
Epoch 45/50
391/391 [==============================] - 2s 4ms/step - loss: 0.6376 - acc:
0.7829
Epoch 46/50
391/391 [==============================] - 2s 4ms/step - loss: 0.6297 - acc:
0.7843
Epoch 47/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6200 - acc:
0.7889
Epoch 48/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6104 - acc:
0.7924
Epoch 49/50
391/391 [==============================] - 1s 4ms/step - loss: 0.6030 - acc:
0.7946
Epoch 50/50
391/391 [==============================] - 1s 4ms/step - loss: 0.5915 - acc:
0.7996
```

# 5. Evaluate the model on the test set (5 points)

Do NOT use the test set until now. Make sure that your model parameters and hyper-parameters are independent of the test set.

In [13]:

```
from keras.models import load_model

current_model = load_model('model_1.h5')
Current_acc = current_model.evaluate(x_test, y_test_vec)
print('loss = ' + str(Current_acc[0]))
print('accuracy = ' + str(Current_acc[1]))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 1.0277 - accur
acy: 0.6602
loss = 1.0277191400527954
accuracy = 0.6601999998092651
```

In [14]:
```
current_model = load_model('model_2.h5')
Current_acc = current_model.evaluate(x_test, y_test_vec)
print('loss = ' + str(Current_acc[0]))
print('accuracy = ' + str(Current_acc[1]))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.8680 - acc:
0.7045
loss = 0.8679858446121216
accuracy = 0.7045000195503235
```

# 6. Building model with new structure (25 points)

- In this section, you can build your model with adding new layers (e.g, BN layer or dropout layer, ...).
- If you want to regularize a `Conv/Dense layer`, you should place a `Dropout layer` before the `Conv/Dense layer`.
- You can try to compare their loss curve and testing accuracy and analyze your findings.
- You need to try at lease two different model structures.

## First Model using Batch Normalization:

In [15]:
```
#Building the model

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (4, 4)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 30, 30, 32)        896       
                                                                 
 batch_normalization (BatchN  (None, 30, 30, 32)       128       
 ormalization)                                                   
                                                                 
 activation (Activation)     (None, 30, 30, 32)        0         
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 15, 15, 32)       0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 12, 12, 64)        32832     
                                                                 
 batch_normalization_1 (Batc  (None, 12, 12, 64)       256       
 hNormalization)                                                 
                                                                 
 activation_1 (Activation)   (None, 12, 12, 64)        0         
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 flatten_2 (Flatten)         (None, 2304)              0         
                                                                 
 dense_4 (Dense)             (None, 256)               590080    
                                                                 
 batch_normalization_2 (Batc  (None, 256)              1024      
 hNormalization)                                                 
                                                                 
 activation_2 (Activation)   (None, 256)               0         
                                                                 
 dense_5 (Dense)             (None, 10)                2570      
                                                                 
=================================================================
Total params: 627,786
Trainable params: 627,082
Non-trainable params: 704
_________________________________________________________________
```

In [16]:
```
# Doing data augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

temp_data = ImageDataGenerator( rotation_range=20, height_shift_range=0.2, wid
```

In [17]:
```
# Define model optimizer and loss function
lr = 0.0001
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(le
```

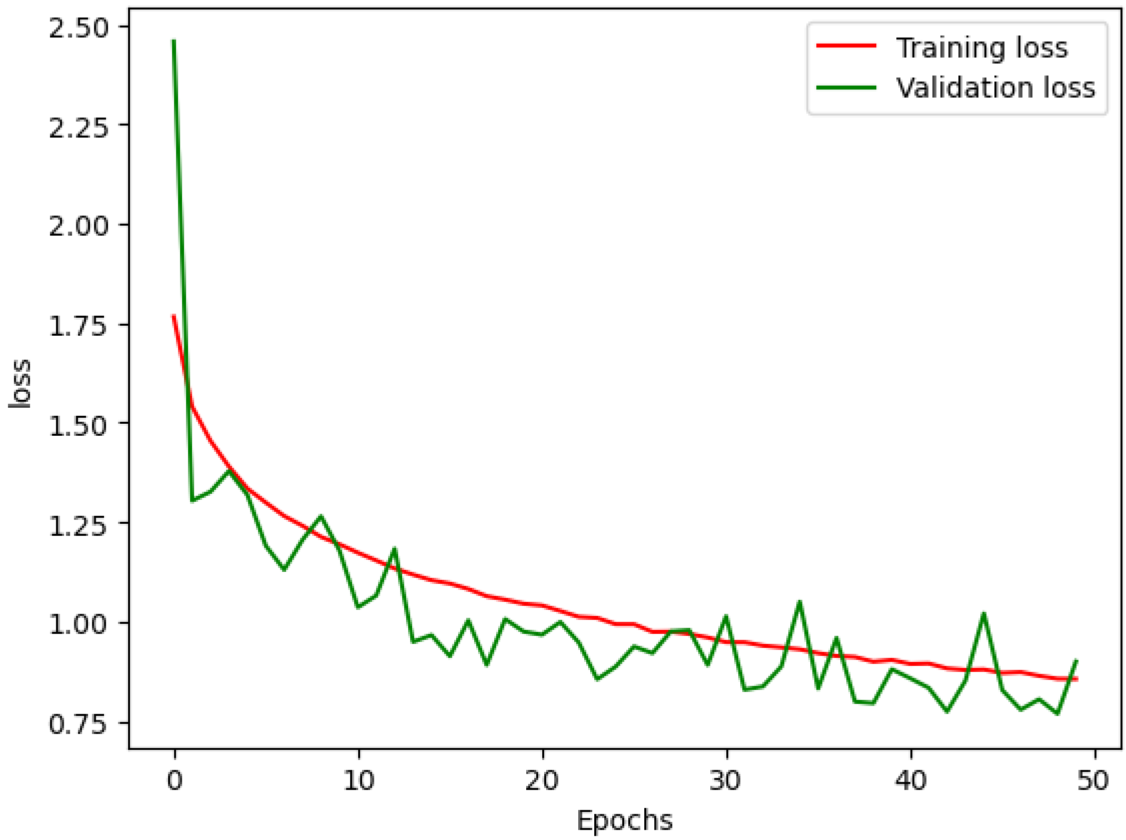
In [18]:
```
# Fits the model with real-time data augmentation

model_3 = model.fit(temp_data.flow(x_tr, y_tr, batch_size=128), steps_per_epoc
model.save('model_3.h5')
```

```
Epoch 1/50
312/312 [==============================] - 14s 41ms/step - loss: 1.7661 - ac
c: 0.3656 - val_loss: 2.4578 - val_acc: 0.1781
Epoch 2/50
312/312 [==============================] - 12s 40ms/step - loss: 1.5402 - ac
c: 0.4496 - val_loss: 1.3038 - val_acc: 0.5356
Epoch 3/50
312/312 [==============================] - 13s 41ms/step - loss: 1.4541 - ac
c: 0.4795 - val_loss: 1.3265 - val_acc: 0.5329
Epoch 4/50
312/312 [==============================] - 13s 40ms/step - loss: 1.3897 - ac
c: 0.5039 - val_loss: 1.3783 - val_acc: 0.5250
Epoch 5/50
312/312 [==============================] - 12s 40ms/step - loss: 1.3344 - ac
c: 0.5272 - val_loss: 1.3185 - val_acc: 0.5375
Epoch 6/50
312/312 [==============================] - 13s 40ms/step - loss: 1.2992 - ac
c: 0.5393 - val_loss: 1.1908 - val_acc: 0.5739
Epoch 7/50
312/312 [==============================] - 13s 41ms/step - loss: 1.2653 - ac
c: 0.5526 - val_loss: 1.1303 - val_acc: 0.6010
Epoch 8/50
312/312 [==============================] - 13s 41ms/step - loss: 1.2407 - ac
c: 0.5639 - val_loss: 1.2059 - val_acc: 0.5753
Epoch 9/50
312/312 [==============================] - 17s 55ms/step - loss: 1.2130 - ac
c: 0.5734 - val_loss: 1.2646 - val_acc: 0.5725
Epoch 10/50
312/312 [==============================] - 12s 40ms/step - loss: 1.1944 - ac
c: 0.5788 - val_loss: 1.1789 - val_acc: 0.5882
Epoch 11/50
312/312 [==============================] - 13s 41ms/step - loss: 1.1733 - ac
c: 0.5857 - val_loss: 1.0359 - val_acc: 0.6362
Epoch 12/50
312/312 [==============================] - 13s 42ms/step - loss: 1.1537 - ac
c: 0.5939 - val_loss: 1.0656 - val_acc: 0.6271
Epoch 13/50
312/312 [==============================] - 13s 41ms/step - loss: 1.1336 - ac
c: 0.6001 - val_loss: 1.1833 - val_acc: 0.5955
Epoch 14/50
312/312 [==============================] - 13s 40ms/step - loss: 1.1182 - ac
c: 0.6079 - val_loss: 0.9492 - val_acc: 0.6704
Epoch 15/50
312/312 [==============================] - 13s 40ms/step - loss: 1.1042 - ac
c: 0.6140 - val_loss: 0.9662 - val_acc: 0.6683
Epoch 16/50
312/312 [==============================] - 13s 41ms/step - loss: 1.0958 - ac
c: 0.6156 - val_loss: 0.9131 - val_acc: 0.6832
Epoch 17/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0821 - ac
c: 0.6196 - val_loss: 1.0033 - val_acc: 0.6528
Epoch 18/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0640 - ac
c: 0.6253 - val_loss: 0.8912 - val_acc: 0.6915
Epoch 19/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0553 - ac
c: 0.6297 - val_loss: 1.0065 - val_acc: 0.6571
```

```
Epoch 20/50
312/312 [==============================] - 13s 42ms/step - loss: 1.0451 - ac
c: 0.6344 - val_loss: 0.9751 - val_acc: 0.6627
Epoch 21/50
312/312 [==============================] - 13s 42ms/step - loss: 1.0408 - ac
c: 0.6318 - val_loss: 0.9671 - val_acc: 0.6597
Epoch 22/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0268 - ac
c: 0.6374 - val_loss: 0.9991 - val_acc: 0.6585
Epoch 23/50
312/312 [==============================] - 13s 41ms/step - loss: 1.0121 - ac
c: 0.6443 - val_loss: 0.9475 - val_acc: 0.6770
Epoch 24/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0093 - ac
c: 0.6469 - val_loss: 0.8552 - val_acc: 0.7046
Epoch 25/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9937 - ac
c: 0.6529 - val_loss: 0.8869 - val_acc: 0.6950
Epoch 26/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9935 - ac
c: 0.6514 - val_loss: 0.9376 - val_acc: 0.6821
Epoch 27/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9744 - ac
c: 0.6583 - val_loss: 0.9209 - val_acc: 0.6819
Epoch 28/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9749 - ac
c: 0.6583 - val_loss: 0.9754 - val_acc: 0.6615
Epoch 29/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9691 - ac
c: 0.6627 - val_loss: 0.9787 - val_acc: 0.6696
Epoch 30/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9602 - ac
c: 0.6633 - val_loss: 0.8911 - val_acc: 0.6914
Epoch 31/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9485 - ac
c: 0.6690 - val_loss: 1.0136 - val_acc: 0.6559
Epoch 32/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9482 - ac
c: 0.6691 - val_loss: 0.8290 - val_acc: 0.7115
Epoch 33/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9397 - ac
c: 0.6729 - val_loss: 0.8371 - val_acc: 0.7097
Epoch 34/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9354 - ac
c: 0.6704 - val_loss: 0.8873 - val_acc: 0.6940
Epoch 35/50
312/312 [==============================] - 13s 42ms/step - loss: 0.9306 - ac
c: 0.6746 - val_loss: 1.0502 - val_acc: 0.6431
Epoch 36/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9209 - ac
c: 0.6783 - val_loss: 0.8319 - val_acc: 0.7164
Epoch 37/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9138 - ac
c: 0.6808 - val_loss: 0.9591 - val_acc: 0.6668
Epoch 38/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9112 - ac
c: 0.6804 - val_loss: 0.7991 - val_acc: 0.7232
```

```
Epoch 39/50
312/312 [==============================] - 13s 41ms/step - loss: 0.8994 - ac
c: 0.6862 - val_loss: 0.7952 - val_acc: 0.7266
Epoch 40/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9039 - ac
c: 0.6818 - val_loss: 0.8805 - val_acc: 0.7011
Epoch 41/50
312/312 [==============================] - 13s 41ms/step - loss: 0.8938 - ac
c: 0.6900 - val_loss: 0.8577 - val_acc: 0.7064
Epoch 42/50
312/312 [==============================] - 13s 41ms/step - loss: 0.8949 - ac
c: 0.6872 - val_loss: 0.8338 - val_acc: 0.7083
Epoch 43/50
312/312 [==============================] - 13s 40ms/step - loss: 0.8825 - ac
c: 0.6917 - val_loss: 0.7739 - val_acc: 0.7294
Epoch 44/50
312/312 [==============================] - 13s 40ms/step - loss: 0.8786 - ac
c: 0.6956 - val_loss: 0.8520 - val_acc: 0.7049
Epoch 45/50
312/312 [==============================] - 13s 40ms/step - loss: 0.8798 - ac
c: 0.6950 - val_loss: 1.0208 - val_acc: 0.6720
Epoch 46/50
312/312 [==============================] - 13s 40ms/step - loss: 0.8711 - ac
c: 0.6958 - val_loss: 0.8281 - val_acc: 0.7132
Epoch 47/50
312/312 [==============================] - 13s 41ms/step - loss: 0.8733 - ac
c: 0.6945 - val_loss: 0.7788 - val_acc: 0.7283
Epoch 48/50
312/312 [==============================] - 12s 40ms/step - loss: 0.8639 - ac
c: 0.6983 - val_loss: 0.8050 - val_acc: 0.7161
Epoch 49/50
312/312 [==============================] - 13s 41ms/step - loss: 0.8565 - ac
c: 0.7012 - val_loss: 0.7685 - val_acc: 0.7353
Epoch 50/50
312/312 [==============================] - 13s 40ms/step - loss: 0.8560 - ac
c: 0.6985 - val_loss: 0.9000 - val_acc: 0.6996
```

In [35]:
```
# Plot the loss curve

loss = model_3.history['loss']
val_loss = model_3.history['val_loss']

plt.xlabel('Epochs')
plt.ylabel('loss')

epochs = range(len(loss))

plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'green', label='Validation loss')

plt.legend()
plt.show()
```




## Train the model on the entire training set

Why? Previously, you used 40K samples for training; you wasted 10K samples for the sake of hyper-parameter tuning. Now you already know the hyper-parameters, so why not using all the 50K samples for training?

In [23]:
```
#<Compile your model again (using the same hyper-parameters you tuned above)>

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (4, 4)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(le

# Data augmentation

temp_data = ImageDataGenerator( rotation_range=20, height_shift_range=0.2, wid
                               zoom_range = 0.2, shear_range = 0.2, horizontal
```

In [24]:
```
model_4 = model.fit(temp_data.flow(x_train, y_train_vec, batch_size=128), step
model.save('model_4.h5')
```

```
Epoch 1/50
312/312 [==============================] - 14s 42ms/step - loss: 1.7751 - ac
c: 0.3651
Epoch 2/50
312/312 [==============================] - 13s 40ms/step - loss: 1.5258 - ac
c: 0.4496
Epoch 3/50
312/312 [==============================] - 13s 41ms/step - loss: 1.4247 - ac
c: 0.4872
Epoch 4/50
312/312 [==============================] - 13s 40ms/step - loss: 1.3614 - ac
c: 0.5133
Epoch 5/50
312/312 [==============================] - 13s 41ms/step - loss: 1.3183 - ac
c: 0.5304
Epoch 6/50
312/312 [==============================] - 13s 41ms/step - loss: 1.2727 - ac
c: 0.5475
Epoch 7/50
312/312 [==============================] - 12s 40ms/step - loss: 1.2425 - ac
c: 0.5579
Epoch 8/50
312/312 [==============================] - 13s 41ms/step - loss: 1.2169 - ac
c: 0.5675
Epoch 9/50
312/312 [==============================] - 13s 41ms/step - loss: 1.1947 - ac
c: 0.5772
Epoch 10/50
312/312 [==============================] - 13s 40ms/step - loss: 1.1706 - ac
c: 0.5853
Epoch 11/50
312/312 [==============================] - 13s 40ms/step - loss: 1.1474 - ac
c: 0.5951
Epoch 12/50
312/312 [==============================] - 13s 41ms/step - loss: 1.1302 - ac
c: 0.6023
Epoch 13/50
312/312 [==============================] - 13s 40ms/step - loss: 1.1183 - ac
c: 0.6049
Epoch 14/50
312/312 [==============================] - 12s 39ms/step - loss: 1.1049 - ac
c: 0.6117
Epoch 15/50
312/312 [==============================] - 13s 41ms/step - loss: 1.0887 - ac
c: 0.6160
Epoch 16/50
312/312 [==============================] - 13s 41ms/step - loss: 1.0790 - ac
c: 0.6201
Epoch 17/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0638 - ac
c: 0.6233
Epoch 18/50
312/312 [==============================] - 13s 41ms/step - loss: 1.0504 - ac
c: 0.6294
Epoch 19/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0440 - ac
c: 0.6320
```

```
Epoch 20/50
312/312 [==============================] - 12s 40ms/step - loss: 1.0279 - ac
c: 0.6377
Epoch 21/50
312/312 [==============================] - 13s 40ms/step - loss: 1.0245 - ac
c: 0.6397
Epoch 22/50
312/312 [==============================] - 12s 40ms/step - loss: 1.0129 - ac
c: 0.6438
Epoch 23/50
312/312 [==============================] - 13s 41ms/step - loss: 1.0065 - ac
c: 0.6457
Epoch 24/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9930 - ac
c: 0.6526
Epoch 25/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9911 - ac
c: 0.6540
Epoch 26/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9862 - ac
c: 0.6552
Epoch 27/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9790 - ac
c: 0.6587
Epoch 28/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9681 - ac
c: 0.6600
Epoch 29/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9621 - ac
c: 0.6640
Epoch 30/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9518 - ac
c: 0.6676
Epoch 31/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9444 - ac
c: 0.6700
Epoch 32/50
312/312 [==============================] - 12s 39ms/step - loss: 0.9454 - ac
c: 0.6674
Epoch 33/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9343 - ac
c: 0.6719
Epoch 34/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9303 - ac
c: 0.6762
Epoch 35/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9282 - ac
c: 0.6753
Epoch 36/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9217 - ac
c: 0.6763
Epoch 37/50
312/312 [==============================] - 12s 39ms/step - loss: 0.9177 - ac
c: 0.6800
Epoch 38/50
312/312 [==============================] - 12s 39ms/step - loss: 0.9100 - ac
c: 0.6822
```

```
Epoch 39/50
312/312 [==============================] - 13s 41ms/step - loss: 0.9041 - ac
c: 0.6855
Epoch 40/50
312/312 [==============================] - 12s 40ms/step - loss: 0.9042 - ac
c: 0.6860
Epoch 41/50
312/312 [==============================] - 13s 40ms/step - loss: 0.9010 - ac
c: 0.6847
Epoch 42/50
312/312 [==============================] - 12s 40ms/step - loss: 0.8935 - ac
c: 0.6879
Epoch 43/50
312/312 [==============================] - 12s 39ms/step - loss: 0.8931 - ac
c: 0.6886
Epoch 44/50
312/312 [==============================] - 12s 39ms/step - loss: 0.8854 - ac
c: 0.6907
Epoch 45/50
312/312 [==============================] - 12s 40ms/step - loss: 0.8883 - ac
c: 0.6902
Epoch 46/50
312/312 [==============================] - 12s 40ms/step - loss: 0.8737 - ac
c: 0.6950
Epoch 47/50
312/312 [==============================] - 12s 40ms/step - loss: 0.8686 - ac
c: 0.6964
Epoch 48/50
312/312 [==============================] - 12s 40ms/step - loss: 0.8716 - ac
c: 0.6942
Epoch 49/50
312/312 [==============================] - 12s 39ms/step - loss: 0.8639 - ac
c: 0.6998
Epoch 50/50
312/312 [==============================] - 12s 39ms/step - loss: 0.8591 - ac
c: 0.7003
```

In [22]:
```
# Evaluate your model performance (testing accuracy) on testing data.

current_model = load_model('model_3.h5')
current_acc = current_model.evaluate(x_test, y_test_vec)
print('loss = ' + str(current_acc[0]))
print('accuracy = ' + str(current_acc[1]))
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.9273 - acc:
0.6935
loss = 0.9273090958595276
accuracy = 0.6934999823570251
```

In [25]:
```
# Evaluate your model performance (testing accuracy) on testing data.

current_model = load_model('model_4.h5')
current_acc = current_model.evaluate(x_test, y_test_vec)
print('loss = ' + str(current_acc[0]))
print('accuracy = ' + str(current_acc[1]))
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.8667 - acc:
0.7015
loss = 0.8666813373565674
accuracy = 0.7014999985694885
```

### Second Model using Batch Normalization and Dropout:

In [26]:
```
# Build the model


model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (4, 4)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_10 (Conv2D)          (None, 30, 30, 32)        896       
                                                                 
 batch_normalization_9 (Batc  (None, 30, 30, 32)       128       
 hNormalization)                                                 
                                                                 
 activation_9 (Activation)   (None, 30, 30, 32)        0         
                                                                 
 max_pooling2d_10 (MaxPoolin  (None, 15, 15, 32)       0         
 g2D)                                                            
                                                                 
 conv2d_11 (Conv2D)          (None, 12, 12, 64)        32832     
                                                                 
 batch_normalization_10 (Bat  (None, 12, 12, 64)       256       
 chNormalization)                                                
                                                                 
 activation_10 (Activation)  (None, 12, 12, 64)        0         
                                                                 
 max_pooling2d_11 (MaxPoolin  (None, 6, 6, 64)         0         
 g2D)                                                            
                                                                 
 flatten_5 (Flatten)         (None, 2304)              0         
                                                                 
 dropout (Dropout)           (None, 2304)              0         
                                                                 
 dense_10 (Dense)            (None, 256)               590080    
                                                                 
 batch_normalization_11 (Bat  (None, 256)              1024      
 chNormalization)                                                
                                                                 
 activation_11 (Activation)  (None, 256)               0         
                                                                 
 dense_11 (Dense)            (None, 10)                2570      
                                                                 
=================================================================
Total params: 627,786
Trainable params: 627,082
Non-trainable params: 704
_________________________________________________________________
```

In [27]:
```
# Define model optimizer and loss function

lr = 0.001
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(le
```

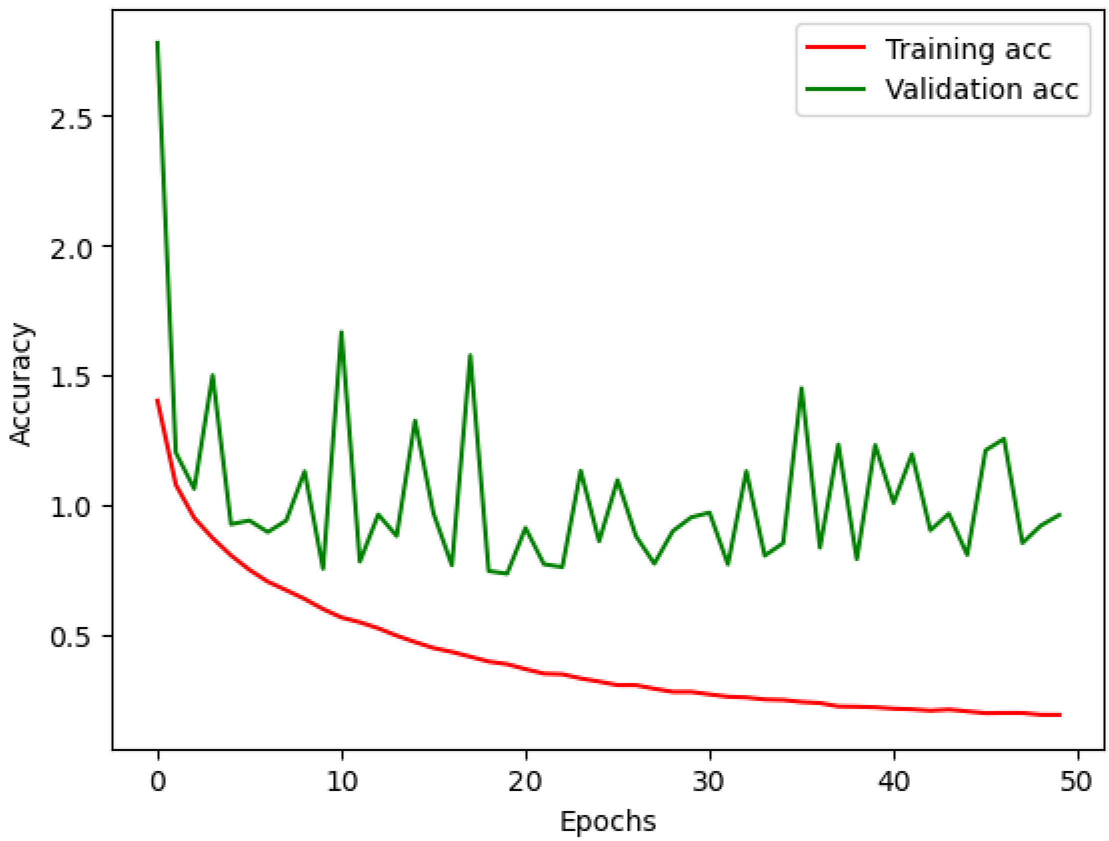
In [28]:
```
# Train the model and store model parameters/loss values

model_5 = model.fit(x_tr, y_tr, batch_size=128, epochs=50, validation_data=(x_
model.save('model_5.h5')
```

```
Epoch 1/50
313/313 [==============================] - 3s 7ms/step - loss: 1.4010 - acc:
0.5006 - val_loss: 2.7804 - val_acc: 0.2018
Epoch 2/50
313/313 [==============================] - 2s 6ms/step - loss: 1.0765 - acc:
0.6166 - val_loss: 1.2008 - val_acc: 0.5772
Epoch 3/50
313/313 [==============================] - 2s 6ms/step - loss: 0.9486 - acc:
0.6665 - val_loss: 1.0610 - val_acc: 0.6245
Epoch 4/50
313/313 [==============================] - 2s 6ms/step - loss: 0.8707 - acc:
0.6932 - val_loss: 1.4986 - val_acc: 0.4899
Epoch 5/50
313/313 [==============================] - 2s 6ms/step - loss: 0.8040 - acc:
0.7161 - val_loss: 0.9244 - val_acc: 0.6689
Epoch 6/50
313/313 [==============================] - 2s 6ms/step - loss: 0.7486 - acc:
0.7391 - val_loss: 0.9386 - val_acc: 0.6749
Epoch 7/50
313/313 [==============================] - 2s 6ms/step - loss: 0.7028 - acc:
0.7531 - val_loss: 0.8941 - val_acc: 0.6939
Epoch 8/50
313/313 [==============================] - 2s 6ms/step - loss: 0.6700 - acc:
0.7646 - val_loss: 0.9386 - val_acc: 0.6734
Epoch 9/50
313/313 [==============================] - 2s 6ms/step - loss: 0.6362 - acc:
0.7772 - val_loss: 1.1281 - val_acc: 0.6478
Epoch 10/50
313/313 [==============================] - 2s 6ms/step - loss: 0.5972 - acc:
0.7930 - val_loss: 0.7531 - val_acc: 0.7419
Epoch 11/50
313/313 [==============================] - 2s 6ms/step - loss: 0.5647 - acc:
0.8009 - val_loss: 1.6648 - val_acc: 0.5407
Epoch 12/50
313/313 [==============================] - 2s 6ms/step - loss: 0.5464 - acc:
0.8089 - val_loss: 0.7800 - val_acc: 0.7381
Epoch 13/50
313/313 [==============================] - 2s 6ms/step - loss: 0.5231 - acc:
0.8166 - val_loss: 0.9620 - val_acc: 0.6889
Epoch 14/50
313/313 [==============================] - 2s 6ms/step - loss: 0.4947 - acc:
0.8278 - val_loss: 0.8791 - val_acc: 0.7092
Epoch 15/50
313/313 [==============================] - 2s 6ms/step - loss: 0.4700 - acc:
0.8353 - val_loss: 1.3240 - val_acc: 0.6039
Epoch 16/50
313/313 [==============================] - 2s 6ms/step - loss: 0.4472 - acc:
0.8436 - val_loss: 0.9667 - val_acc: 0.6909
Epoch 17/50
313/313 [==============================] - 2s 6ms/step - loss: 0.4319 - acc:
0.8504 - val_loss: 0.7658 - val_acc: 0.7568
Epoch 18/50
313/313 [==============================] - 2s 6ms/step - loss: 0.4136 - acc:
0.8548 - val_loss: 1.5756 - val_acc: 0.6043
Epoch 19/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3949 - acc:
0.8626 - val_loss: 0.7440 - val_acc: 0.7624
```

```
Epoch 20/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3848 - acc:
0.8670 - val_loss: 0.7336 - val_acc: 0.7613
Epoch 21/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3656 - acc:
0.8736 - val_loss: 0.9102 - val_acc: 0.7213
Epoch 22/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3485 - acc:
0.8770 - val_loss: 0.7705 - val_acc: 0.7434
Epoch 23/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3459 - acc:
0.8797 - val_loss: 0.7592 - val_acc: 0.7550
Epoch 24/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3299 - acc:
0.8838 - val_loss: 1.1308 - val_acc: 0.6733
Epoch 25/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3176 - acc:
0.8882 - val_loss: 0.8588 - val_acc: 0.7492
Epoch 26/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3041 - acc:
0.8938 - val_loss: 1.0947 - val_acc: 0.7022
Epoch 27/50
313/313 [==============================] - 2s 6ms/step - loss: 0.3039 - acc:
0.8953 - val_loss: 0.8790 - val_acc: 0.7471
Epoch 28/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2901 - acc:
0.9004 - val_loss: 0.7724 - val_acc: 0.7734
Epoch 29/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2781 - acc:
0.9031 - val_loss: 0.8976 - val_acc: 0.7265
Epoch 30/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2780 - acc:
0.9041 - val_loss: 0.9509 - val_acc: 0.7280
Epoch 31/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2684 - acc:
0.9074 - val_loss: 0.9693 - val_acc: 0.7400
Epoch 32/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2594 - acc:
0.9100 - val_loss: 0.7696 - val_acc: 0.7632
Epoch 33/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2566 - acc:
0.9105 - val_loss: 1.1291 - val_acc: 0.6982
Epoch 34/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2488 - acc:
0.9149 - val_loss: 0.8032 - val_acc: 0.7610
Epoch 35/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2468 - acc:
0.9153 - val_loss: 0.8516 - val_acc: 0.7659
Epoch 36/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2392 - acc:
0.9173 - val_loss: 1.4489 - val_acc: 0.6315
Epoch 37/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2349 - acc:
0.9196 - val_loss: 0.8341 - val_acc: 0.7567
Epoch 38/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2220 - acc:
0.9231 - val_loss: 1.2317 - val_acc: 0.6546
```

```
Epoch 39/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2209 - acc:
0.9238 - val_loss: 0.7900 - val_acc: 0.7576
Epoch 40/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2186 - acc:
0.9247 - val_loss: 1.2296 - val_acc: 0.6916
Epoch 41/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2134 - acc:
0.9261 - val_loss: 1.0066 - val_acc: 0.7182
Epoch 42/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2108 - acc:
0.9265 - val_loss: 1.1948 - val_acc: 0.6981
Epoch 43/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2058 - acc:
0.9285 - val_loss: 0.9009 - val_acc: 0.7587
Epoch 44/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2100 - acc:
0.9266 - val_loss: 0.9652 - val_acc: 0.7536
Epoch 45/50
313/313 [==============================] - 2s 6ms/step - loss: 0.2030 - acc:
0.9302 - val_loss: 0.8061 - val_acc: 0.7746
Epoch 46/50
313/313 [==============================] - 2s 6ms/step - loss: 0.1959 - acc:
0.9326 - val_loss: 1.2095 - val_acc: 0.7192
Epoch 47/50
313/313 [==============================] - 2s 6ms/step - loss: 0.1968 - acc:
0.9323 - val_loss: 1.2535 - val_acc: 0.6457
Epoch 48/50
313/313 [==============================] - 2s 6ms/step - loss: 0.1967 - acc:
0.9325 - val_loss: 0.8520 - val_acc: 0.7464
Epoch 49/50
313/313 [==============================] - 2s 6ms/step - loss: 0.1897 - acc:
0.9346 - val_loss: 0.9192 - val_acc: 0.7530
Epoch 50/50
313/313 [==============================] - 2s 6ms/step - loss: 0.1895 - acc:
0.9351 - val_loss: 0.9600 - val_acc: 0.7219
```

In [34]:
```
# Plot the loss curve

import matplotlib.pyplot as plt
%matplotlib inline

loss = model_5.history['loss']
val_loss = model_5.history['val_loss']

epochs = range(len(loss))

plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.plot(epochs, loss, 'red', label='Training acc')
plt.plot(epochs, val_loss, 'green', label='Validation acc')

plt.legend()
plt.show()
```




## Train the model on the entire training set

Why? Previously, you used 40K samples for training; you wasted 10K samples for the sake of hyper-parameter tuning. Now you already know the hyper-parameters, so why not using all the 50K samples for training?

In [30]:
```
#<Compile your model again (using the same hyper-parameters you tuned above)>

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Activa
from keras.models import Sequential

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (4, 4)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(le
```

In [31]:
```
# Train the model and store model parameters/loss values

model_6 = model.fit(x_train, y_train_vec, batch_size=128, epochs=50)
model.save('model_6.h5')
```

```
Epoch 1/50
391/391 [==============================] - 3s 6ms/step - loss: 1.3297 - acc:
0.5261
Epoch 2/50
391/391 [==============================] - 2s 6ms/step - loss: 1.0174 - acc:
0.6375
Epoch 3/50
391/391 [==============================] - 2s 6ms/step - loss: 0.8976 - acc:
0.6840
Epoch 4/50
391/391 [==============================] - 2s 6ms/step - loss: 0.8171 - acc:
0.7147
Epoch 5/50
391/391 [==============================] - 2s 6ms/step - loss: 0.7628 - acc:
0.7312
Epoch 6/50
391/391 [==============================] - 2s 6ms/step - loss: 0.7103 - acc:
0.7508
Epoch 7/50
391/391 [==============================] - 2s 6ms/step - loss: 0.6668 - acc:
0.7684
Epoch 8/50
391/391 [==============================] - 2s 6ms/step - loss: 0.6324 - acc:
0.7792
Epoch 9/50
391/391 [==============================] - 2s 6ms/step - loss: 0.5946 - acc:
0.7924
Epoch 10/50
391/391 [==============================] - 2s 6ms/step - loss: 0.5733 - acc:
0.7995
Epoch 11/50
391/391 [==============================] - 2s 6ms/step - loss: 0.5407 - acc:
0.8099
Epoch 12/50
391/391 [==============================] - 2s 6ms/step - loss: 0.5228 - acc:
0.8190
Epoch 13/50
391/391 [==============================] - 2s 6ms/step - loss: 0.4952 - acc:
0.8270
Epoch 14/50
391/391 [==============================] - 2s 6ms/step - loss: 0.4778 - acc:
0.8338
Epoch 15/50
391/391 [==============================] - 2s 6ms/step - loss: 0.4559 - acc:
0.8417
Epoch 16/50
391/391 [==============================] - 2s 6ms/step - loss: 0.4435 - acc:
0.8457
Epoch 17/50
391/391 [==============================] - 2s 6ms/step - loss: 0.4250 - acc:
0.8520
Epoch 18/50
391/391 [==============================] - 2s 6ms/step - loss: 0.4097 - acc:
0.8567
Epoch 19/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3935 - acc:
0.8624
```

```
Epoch 20/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3785 - acc:
0.8679
Epoch 21/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3661 - acc:
0.8713
Epoch 22/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3500 - acc:
0.8781
Epoch 23/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3450 - acc:
0.8799
Epoch 24/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3376 - acc:
0.8826
Epoch 25/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3224 - acc:
0.8870
Epoch 26/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3154 - acc:
0.8895
Epoch 27/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3101 - acc:
0.8910
Epoch 28/50
391/391 [==============================] - 2s 6ms/step - loss: 0.3002 - acc:
0.8956
Epoch 29/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2928 - acc:
0.8983
Epoch 30/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2854 - acc:
0.9002
Epoch 31/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2811 - acc:
0.9013
Epoch 32/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2698 - acc:
0.9057
Epoch 33/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2688 - acc:
0.9061
Epoch 34/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2648 - acc:
0.9062
Epoch 35/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2544 - acc:
0.9115
Epoch 36/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2522 - acc:
0.9121
Epoch 37/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2430 - acc:
0.9162
Epoch 38/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2475 - acc:
0.9130
```

```
Epoch 39/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2404 - acc:
0.9152
Epoch 40/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2332 - acc:
0.9191
Epoch 41/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2311 - acc:
0.9202
Epoch 42/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2256 - acc:
0.9204
Epoch 43/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2244 - acc:
0.9224
Epoch 44/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2187 - acc:
0.9245
Epoch 45/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2201 - acc:
0.9237
Epoch 46/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2133 - acc:
0.9254
Epoch 47/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2125 - acc:
0.9271
Epoch 48/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2111 - acc:
0.9273
Epoch 49/50
391/391 [==============================] - 2s 6ms/step - loss: 0.2081 - acc:
0.9264
Epoch 50/50
391/391 [==============================] - 2s 5ms/step - loss: 0.2022 - acc:
0.9284
```

In [32]:

```
# Evaluate your model performance (testing accuracy) on testing data.

current_model = load_model('model_5.h5')
current_acc = current_model.evaluate(x_test, y_test_vec)
print('loss = ' + str(current_acc[0]))
print('accuracy = ' + str(current_acc[1]))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.9862 - acc:
0.7103
loss = 0.9862155318260193
accuracy = 0.7103000283241272
```

In [33]:
```
# Evaluate your model performance (testing accuracy) on testing data.

current_model = load_model('model_6.h5')
current_acc = current_model.evaluate(x_test, y_test_vec)
print('loss = ' + str(current_acc[0]))
print('accuracy = ' + str(current_acc[1]))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 1.1104 - acc:
0.7182
loss = 1.1104212999343872
accuracy = 0.7182000279426575
```

# Comparision and Analysis:

*Normal CNN model:* is not that efficient and has an accuracy of 66 % and after using full training dataset it achieves accuracy of 70 %. *My first model* which uses data augmentation and batch normalization has a accuracy of 69 % and after using full training dataset it achieves has accuracy of 70 %. *My second and final model* which uses batch normaliztion and dropout has an accuracy of 71 % and after using full training dataset it achieves an accuracy of ~72 %.

In [36]:
```
from numba import cuda
device = cuda.get_current_device()
device.reset()
```