Keras; Convolutional Neural Network (CNN); ten-class classifier for CIFAR-10 dataset

Link: https://colab.research.google.com/drive/10VU59wS6R9N2_hCuDSBzZclTateh5VYx?usp=sharing

Libraries

```
from keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np
from random import randint
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNorm
import tensorflow as tf
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score
from keras.optimizers import Adam
from keras.utils import to_categorical
```

Defintions

```
def img_plt(images, labels):
    plt.figure(figsize=(15, 8))
    for i in range(1, 11):
        plt.subplot(2, 5, i)
        plt.imshow(images[i-1, :, :, :])
        plt.title('Label: ' + str(labels[i-1]))
    plt.show()

def feat_plot(features, labels, classes):
    for class_i in classes:
        plt.plot(features[labels == class_i][:, 0], features[labels == class_i][:
    #plt.axis([-2.2, 2.2, -2.2])
    plt.xlabel('x: feature i')
    plt.ylabel('y: feature 2')
    plt.legend(['Class' + str(classes[class_i]) for class_i in classes])
    plt.show()

def acc_fun(labels_actual, labels_pred):
    acc = np.sum(labels_actual == labels_pred) / len(labels_actual) * 100
    return acc

def plot_curve(accuracy_train, loss_train, accuracy_val, loss_val):
    epochs = np.arange(loss_train.shape[0])
    plt.subplot(1, 2, 1)
    plt.plot(epochs, accuracy_train, epochs, accuracy_val)
    #plt.axis([1,2,1,2])
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy')
    plt.legend(['Training', 'Validation'])

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss_train, epochs, loss_val)
    plt.xlabel('Epoch')
    plt.ylabel('Binary crossentropy loss')
    plt.title('Loss')
    plt.legend(['Training', 'Validation'])
    plt.show()
```

A. Use cifar10 function in keras.datasets to load CIFAR-10 dataset. Split it into the training and testing sets. Define a validation set by randomly selecting 20% of the training images along with their corresponding labels. This will be the "validation_data".

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
classes=np.arange(10)
print(x_train.shape)
num_train_img=x_train.shape[0]
train_ind=np.arange(0,num_train_img)
train_ind_s=np.random.permutation(train_ind)
x_train=x_train[train_ind_s,:,:,:]
y_train=y_train[train_ind_s]
x_val=x_train[0:int(0.2*num_train_img),:,:,:]
y_val=y_train[0:int(0.2*num_train_img)]
x_train=x_train[int(0.2*num_train_img):,:,:]
y_train=y_train[int(0.2*num_train_img):]
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
(50000, 32, 32, 3)
```

B. Scale the pixel values of the images in all the sets to a value between 0 and 1. Perform this process by dividing the image values with 255. Note: No need to flatten the images.

```
x_train=x_train.astype('float32')
x_val=x_val.astype('float32')
x_test=x_test.astype('float32')
x_train /= 255
x_val /= 255
x_test /= 255
```

C. Convert the label vectors for all the sets to binary class matrices using to_categorical() Keras function.

```
y_train_c = to_categorical(y_train, len(classes))
y_val_c = to_categorical(y_val, len(classes))
y_test_c = to_categorical(y_test, len(classes))
```

D. Using Keras library, build a CNN with the following design: 2 convolutional blocks, 1 flattening layer,1 FC layer with 512 nodes, and 1output layer. Each convolutional block consists of two back-to-back Conv layers followed by max pooling. The filter size is 3x3x image_depth. The number of filters is 32 in the first convolutional block and 64 in the second block. Use the following network architecture as a reference:

```
model_a = Sequential()

model_a.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=x_t
model_a.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model_a.add(MaxPooling2D(pool_size=(2, 2)))

model_a.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model_a.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model_a.add(MaxPooling2D(pool_size=(2, 2)))

model_a.add(Flatten())

model_a.add(Dense(512, activation='relu'))

model_a.add(Dropout(0.5))

model_a.add(Dense(len(classes), activation='softmax'))

model_a.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

model_a.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0
 D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 512)               2097664

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 2168362 (8.27 MB)
Trainable params: 2168362 (8.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

E. Compile, train, and then evaluate: Compile the network. Make sure to select a correct loss function for this classification problem. Use Adam optimizer (Adam, learning rate of 0.001). Use ModelCheckpoint to save the best model based on the lowest validation loss. Train the network for 50 epochs with a batch size of 32. Remember to assign the validation set to validation_data in the fit function. Plot the training and validation loss for all the epochs in one plot. Use the evaluate() Keras function to find the training and validation loss and the accuracy. Report the results.

```
# Create a checkpoint to save the model based on the lowest validation loss.
save_path = '/content/drive/My Drive/JennnaLealiA12.h5'
callbacks_save = ModelCheckpoint(save_path, monitor='val_loss', verbose=0, save_b
```

```
# Compile the model with Adam optimizer and a learning rate of 0.001
model_a.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model_a.fit(x_train, y_train_c,
                      batch_size=32,
                      epochs=50,
                      verbose=1,
                      validation_data=(x_val, y_val_c),
                      callbacks=[callbacks_save])

# Plot the training and validation accuracy and loss
plt.figure(figsize=(9,5))
acc_curve_train = np.array(history.history['accuracy'])
loss_curve_train = np.array(history.history['loss'])
acc_curve_val = np.array(history.history['val_accuracy'])
loss_curve_val = np.array(history.history['val_loss'])
plot_curve(acc_curve_train, loss_curve_train, acc_curve_val, loss_curve_val)

# Load the best model saved based on the lowest validation loss
model_a = load_model(save_path)
```

```
Epoch 1/50
1250/1250 [==============================] - 288s 228ms/step - loss: 1.4938 -
Epoch 2/50
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: Use
  saving_api.save_model(
1250/1250 [==============================] - 253s 203ms/step - loss: 1.0316 -
Epoch 3/50
1250/1250 [==============================] - 252s 202ms/step - loss: 0.8511 -
Epoch 4/50
1250/1250 [==============================] - 237s 190ms/step - loss: 0.7213 -
Epoch 5/50
1250/1250 [==============================] - 237s 190ms/step - loss: 0.6228 -
Epoch 6/50
1250/1250 [==============================] - 233s 186ms/step - loss: 0.5304 -
Epoch 7/50
1250/1250 [==============================] - 234s 187ms/step - loss: 0.4513 -
Epoch 8/50
1250/1250 [==============================] - 228s 183ms/step - loss: 0.3861 -
Epoch 9/50
1250/1250 [==============================] - 237s 190ms/step - loss: 0.3351 -
Epoch 10/50
1250/1250 [==============================] - 242s 194ms/step - loss: 0.2925 -
Epoch 11/50
1250/1250 [==============================] - 236s 189ms/step - loss: 0.2667 -
Epoch 12/50
1250/1250 [==============================] - 228s 183ms/step - loss: 0.2396 -
Epoch 13/50
1250/1250 [==============================] - 236s 189ms/step - loss: 0.2307 -
Epoch 14/50
1250/1250 [==============================] - 231s 185ms/step - loss: 0.2037 -
Epoch 15/50
1234/1250 [============================>.] - ETA: 2s - loss: 0.1927 - accuracy
```

```
# Evaluating the model on the training samples
score = model_a.evaluate(x_train, y_train_c)
print('Total loss on training set: ', score[0])
print('Accuracy of training set: ', score[1])

# Evaluating the model on the validation samples
score = model_a.evaluate(x_val, y_val_c)
print('Total loss on validation set: ', score[0])
print('Accuracy of validation set: ', score[1])
```

```
1250/1250 [==============================] - 5s 4ms/step - loss: 0.3957 - accu
Total loss on training set:  0.3956603407859802
Accuracy of training set:  0.8710749745368958
313/313 [==============================] - 1s 3ms/step - loss: 0.7277 - accura
Total loss on validation set:  0.7277424335479736
Accuracy of validation set:  0.745199978351593
```

F. Now define another model with the same architecture in (d) and then: Compile the network. Make sure to select a correct loss function for this classification problem. Use Adam optimizer (Adam, learning rate of 0.001). Use ModelCheckpoint to save the best model based on the lowest validation loss. Create an image data generator in Keras for real-time data augmentation. The augmentation operations are rotation (10 degrees range), width and height shift (0.1 range), and horizontal flip. Train the network for 50 epochs with a batch size of 32. Remember to use the image data generator. Assign the validation set to validation_data in the fit function. Plot the training and validation loss for all the epochs in one plot. Use the evaluate() Keras function to find the training and validation loss and the accuracy. Report the results.

```
# Define a new model with the same architecture
model_b = Sequential()

model_b.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=x_t
model_b.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model_b.add(MaxPooling2D(pool_size=(2, 2)))
model_b.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model_b.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model_b.add(MaxPooling2D(pool_size=(2, 2)))
model_b.add(Flatten())
model_b.add(Dense(512, activation='relu'))
model_b.add(Dropout(0.5))
```

```
model_b.add(Dense(len(classes), activation='softmax'))

model_b.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

model_b.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 32)        896

 conv2d_5 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_7 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 flatten_1 (Flatten)         (None, 4096)              0

 dense_2 (Dense)             (None, 512)               2097664

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 2168362 (8.27 MB)
Trainable params: 2168362 (8.27 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from keras.preprocessing.image import ImageDataGenerator

# Checkpoint to save the model based on the lowest validation loss
save_path_b = '/content/drive/My Drive/JennnaLealiA12_model_b.h5'
callbacks_save_b = ModelCheckpoint(save_path_b, monitor='val_loss', verbose=0, sa

# Create an image data generator for real-time data augmentation
```

```
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

# Fit the model using the image data generator
history_b = model_b.fit(
    datagen.flow(x_train, y_train_c, batch_size=32),
    epochs=50,
    verbose=1,
    validation_data=(x_val, y_val_c),
    callbacks=[callbacks_save_b]
)

# Plot the training and validation accuracy and loss
plt.figure(figsize=(9,5))
acc_curve_train_b = np.array(history_b.history['accuracy'])
loss_curve_train_b = np.array(history_b.history['loss'])
acc_curve_val_b = np.array(history_b.history['val_accuracy'])
loss_curve_val_b = np.array(history_b.history['val_loss'])
plot_curve(acc_curve_train_b, loss_curve_train_b, acc_curve_val_b, loss_curve_val

# Load the best model saved based on the lowest validation loss
model_b = load_model(save_path_b)

# Evaluate the model on the training samples
score_train_b = model_b.evaluate(x_train, y_train_c)
print('Total loss on training set: ', score_train_b[0])
print('Accuracy of training set: ', score_train_b[1])

# Evaluate the model on the validation samples
score_val_b = model_b.evaluate(x_val, y_val_c)
print('Total loss on validation set: ', score_val_b[0])
print('Accuracy of validation set: ', score_val_b[1])
```

```
Epoch 1/50
1250/1250 [==============================] - 33s 25ms/step - loss: 1.5616 - ac
Epoch 2/50
1250/1250 [==============================] - 29s 23ms/step - loss: 1.2006 - ac
Epoch 3/50
1250/1250 [==============================] - 29s 23ms/step - loss: 1.0737 - ac
Epoch 4/50
1250/1250 [==============================] - 29s 23ms/step - loss: 0.9901 - ac
Epoch 5/50
1250/1250 [==============================] - 30s 24ms/step - loss: 0.9270 - ac
Epoch 6/50
1250/1250 [==============================] - 29s 23ms/step - loss: 0.8833 - ac
Epoch 7/50
1250/1250 [==============================] - 28s 23ms/step - loss: 0.8450 - ac
Epoch 8/50
1250/1250 [==============================] - 30s 24ms/step - loss: 0.8105 - ac
Epoch 9/50
1250/1250 [==============================] - 28s 23ms/step - loss: 0.7860 - ac
Epoch 10/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.7631 - ac
Epoch 11/50
1250/1250 [==============================] - 29s 23ms/step - loss: 0.7479 - ac
Epoch 12/50
1250/1250 [==============================] - 27s 22ms/step - loss: 0.7379 - ac
Epoch 13/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.7210 - ac
Epoch 14/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.7005 - ac
Epoch 15/50
1250/1250 [==============================] - 27s 22ms/step - loss: 0.6985 - ac
Epoch 16/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.6818 - ac
Epoch 17/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.6734 - ac
Epoch 18/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.6661 - ac
Epoch 19/50
1250/1250 [==============================] - 28s 22ms/step - loss: 0.6633 - ac
Epoch 20/50
1250/1250 [==============================] - 29s 23ms/step - loss: 0.6539 - ac
Epoch 21/50
 594/1250 [=============>................] - ETA: 14s - loss: 0.6317 - accurac
```

G. What do you observe from the validation loss in both step (e) and (f)? Is the model overfitting or underfitting the training data? Explain.

In model a from step e the validation loss decreases and then starts to increase after a certain number of epochs, which is an indicator of overfitting. Also for model a, the accuracy on the training data is quite high compared to the validation accuracy, which could mean overfitting. In model b from step f the validation loss decreases with the training loss and remains close to it throughout the training process. It does show some changes but does not diverge, meaning that the model with data augmentation from step f generalizes better than the model from step e. Also the validation accuracy is also closer to the training accuracy compared to model a from step e, which could mean that the data augmentation has helped reduce overfitting.

H. Now define another model with the same architecture in (d), except that this time you need to add batch normalization layers to the CNN network. Add normalization layer after all the convolutional and fully connected layers (not the output layer). Add them before the activation layers and be noted that there is no need for the bias in the convolutional or fully connected layers. Compile the network. Make sure to select a correct loss function for this classification problem. Use Adam optimizer (Adam, learning rate of 0.01). Use ModelCheckpoint to save the best model based on the lowest validation loss. Train the network for 50 epochs with a batch size of 64. Remember to assign the validation set to validation_data in the fit function. Plot the training and validation loss for all the epochs in one plot. Use the evaluate() Keras function to find the training and validation loss and

```
from keras.layers import BatchNormalization

model_c = Sequential()

# First Convolutional Block
model_c.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:], use_b
model_c.add(BatchNormalization())
model_c.add(Activation('relu'))
model_c.add(Conv2D(32, (3, 3), padding='same', use_bias=False))
model_c.add(BatchNormalization())
model_c.add(Activation('relu'))
model_c.add(MaxPooling2D(pool_size=(2, 2)))

# Second Convolutional Block
model_c.add(Conv2D(64, (3, 3), padding='same', use_bias=False))
model_c.add(BatchNormalization())
model_c.add(Activation('relu'))
model_c.add(Conv2D(64, (3, 3), padding='same', use_bias=False))
```

```
model_c.add(BatchNormalization())
model_c.add(Activation('relu'))
model_c.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening Layer
model_c.add(Flatten())

# Fully Connected Layer
model_c.add(Dense(512, use_bias=False))
model_c.add(BatchNormalization())
model_c.add(Activation('relu'))

# Output Layer
model_c.add(Dropout(0.5))
model_c.add(Dense(len(classes), activation='softmax'))

# Compile the model with Adam optimizer and a learning rate of 0.01
model_c.compile(optimizer=Adam(learning_rate=0.01),
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# Checkpoint to save the model based on the lowest validation loss
save_path_c = '/content/drive/My Drive/JennnaLealiA12_model_c.h5'
callbacks_save_c = ModelCheckpoint(save_path_c, monitor='val_loss', verbose=0, save

history_c = model_c.fit(x_train, y_train_c,
                        batch_size=64,
                        epochs=50,
                        verbose=1,
                        validation_data=(x_val, y_val_c),
                        callbacks=[callbacks_save_c])

plt.figure(figsize=(9,5))
acc_curve_train_c = np.array(history_c.history['accuracy'])
loss_curve_train_c = np.array(history_c.history['loss'])
acc_curve_val_c = np.array(history_c.history['val_accuracy'])
loss_curve_val_c = np.array(history_c.history['val_loss'])
plot_curve(acc_curve_train_c, loss_curve_train_c, acc_curve_val_c, loss_curve_val_c

# Load the best model saved based on the lowest validation loss
model_c = load_model(save_path_c)

score_train_c = model_c.evaluate(x_train, y_train_c)
print('Total loss on training set: ', score_train_c[0])
print('Accuracy of training set: ', score_train_c[1])
```

```
score_val_c = model_c.evaluate(x_val, y_val_c)
print('Total loss on validation set: ', score_val_c[0])
print('Accuracy of validation set: ', score_val_c[1])
```

I. What do you observe from the training loss in both steps (e) and (h)? Explain.

In step e, the model b's training loss drops consistently, showing it's learning from the data. When I added batch normalization in step h, the loss goes down even more smoothly and ends up lower, meaning model c's learning has improved. Batch normalization helps the model in step h train more reliably and possibly quicker. This makes it easier for the model to adjust and fine-tune during training. Also, in step h, the training loss decreases to a lower plateau compared to step e, which is a sign of effective learning, and it could indicate better generalization due to the regularizing effect of batch normalization.