Submitted by : SAI KRISHNA W.D.S
Unity Id: swupadr

ECE 763 COMPUTER VISION PROJECT -3 REPORT

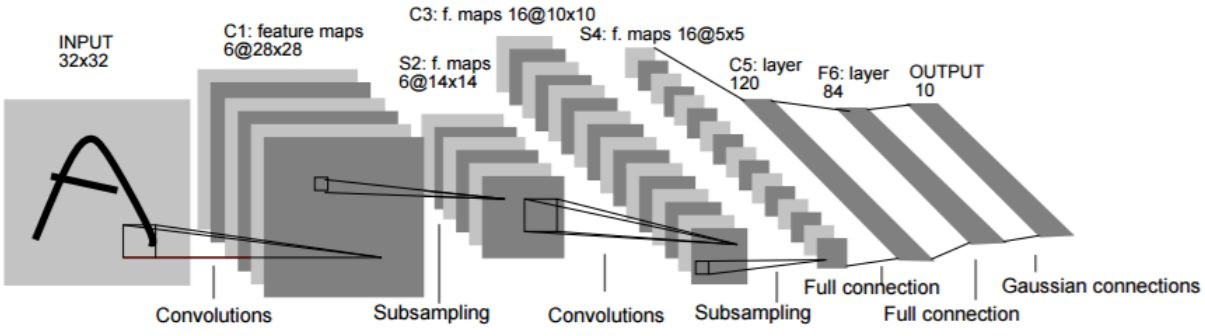
TASK: Babysitting the process of training of Deep Neural Networks for the CIFAR dataset, and testing using the face and non-face data used in an earlier project (project 1).

IMAGE DATA USED:

- I have used the CIFAR -10 for demonstrating the babysitting procedure and Face-NonFace dataset for demonstrating the babysitting procedure and testing the code's performance.
- The CIFAR 10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- The 10 different classes for the CIFAR 10 dataset are {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}
- The Face-NonFace dataset was manually extracted by me from the FDDB faces dataset- as 1000 images for faces, and 1000 images for NonFace. 200 images have been used for testing- 100 each for face and Nonface. For training and testing, the images were cropped to dimensions of 128x128x3.

ARCHITECTURE OF NEURAL NETWORK USED:

I have performed all operations needed for testing and babysitting on top of the LeNeT-5 convolutional neural network. LeNet-5 has all the basic units of a convolutional neural network such as convolutional layer, pooling layer and full connection layer, laying a foundation for the future development of convolutional neural network. As shown in the figure (input image data with 32*32 pixels) : lenet-5 consists of seven layers. In addition to input, every other layer can train parameters.



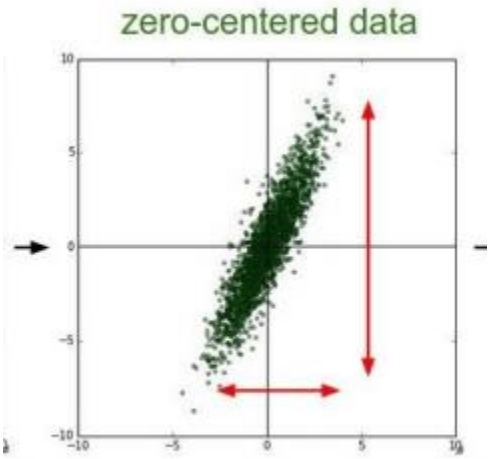
Cx represents convolution layer, Sx represents sub-sampling layer, Fx represents complete connection layer, and x represents layer index.

Submitted by : SAI KRISHNA W.D.S
Unity Id: swupadr

DATA PRE-PROCESSING AND DATA AUGMENTATION:

Data pre-processing involves quality representation of data for easier analysis. In this project I have used the following measures for data pre-processing:

1. **Zero-centering the data distribution**:
   I am using a matrix X which contains all the images, and then obtaining the mean of all the images in X, and then subtracting the mean image from all the images in X. This ensures that the distribution is now centered around 0.

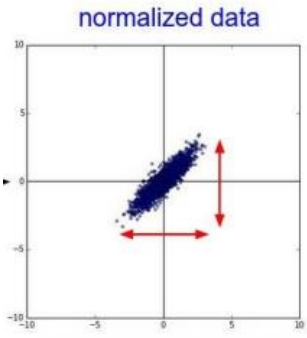



```
X -= np.mean(X, axis = 0)
```

Code snippet:

```
xTrain = xTrain.astype('float32')
xTest = xTest.astype('float32')
xTrain /= 255
xTest /= 255
xTrain -= np.mean(xTrain)
xTest -= np.mean(xTest)
```

2. **Data Normalization:**
   The same matrix X is used to obtain the standard deviation of the image vectors in X, which is then subtracted from all the images in X- to obtain normalized data.




```
X /= np.std(X, axis = 0)
```

Submitted by : SAI KRISHNA W.D.S
Unity Id: swupadr

Code Snippet:

```
xTrain = xTrain.astype('float32')
xTest = xTest.astype('float32')
xTrain /= 255
xTest /= 255
xTrain -= np.mean(xTrain)
xTest -= np.mean(xTest)
xTrain /= np.std(xTrain, axis = 0)
xTest /= np.std(xTest, axis = 0)

model = LeNet((xTrain[0].shape), num_classes)
EPOCHS = 10

H = model.fit(xTrain, yTrain,
    validation_data=(xTest, yTest),
    epochs=EPOCHS, verbose=1)
```

DATA AUGMENTATION :

```
# uncomment the following for data-augmentation:

aug = ImageDataGenerator(rotation_range=35, width_shift_range=0.1,
            height_shift_range=0.1,zoom_range=0.3,shear_range=0.1, fill_mode="reflect")

        valaug = ImageDataGenerator()

        model = LeNet((xTrain[0].shape), num_classes,lr,reg)
        EPOCHS = 20
        H = model.fit_generator(aug.flow(xTrain, yTrain, batch_size = 8),
                    validation_data=valaug.flow(xTest, yTest),
                    epochs=EPOCHS, verbose=1)

# COMMENT THE FOLLOWING LINES AND JUST RUN THE ALREADY COMMENTED (xTRAIN,yTRAIN),(xTEST,yTEST) line below for obtainig v

for count in range(2, 3):
    for iteration in range (0,10):
        iteration = iteration+1
        keras.backend.clear_session()
        #(xTrain,yTrain),(xTest,yTest) = datasets.cifar10.load_data()
        reg = random()*pow(10,np.random.randint(-5, 0))
        lr = random()*pow(10,-count)
        print("Learning rate: ",lr, "Regularisation", reg)
```

WEIGHT INITIALIZATION USING XAVIER :

```
opt=Adam(learning_rate=10**-3)
class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()

        self.add(Conv2D(6, kernel_initializer='glorot_uniform', kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape, padd
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Conv2D(16, kernel_initializer='glorot_uniform', kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='valid')) # tanh cha
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, kernel_initializer='glorot_uniform', activation='relu',activity_regularizer=regularizers.l2(2e-3)))
        self.add(Dense(84, kernel_initializer='glorot_uniform', activation='relu',activity_regularizer=regularizers.l2(2e-3)))
        self.add(Dense(nb_classes, activation='softmax'))

        self.compile(optimizer=opt,loss=categorical_crossentropy,metrics=['accuracy'])
```

Xavier weight initialization given as kernel_initializer=’glorot_uniform’

The following are the o/p values on training after using Xavier:

```
In [19]: runfile('D:/SEM 2-spring 2020/COMPUTER VISION/project-3/CVTF.py', wdir='D:/SEM 2-spring 2020
COMPUTER VISION/project-3')
Reloaded modules: network
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 9s 179us/sample - loss: 1.5559 - accuracy: 0.4551 -
val_loss: 1.3288 - val_accuracy: 0.5384
Epoch 2/10
50000/50000 [==============================] - 8s 167us/sample - loss: 1.2434 - accuracy: 0.5728 -
val_loss: 1.1863 - val_accuracy: 0.5877
Epoch 3/10
50000/50000 [==============================] - 8s 165us/sample - loss: 1.1079 - accuracy: 0.6196 -
val_loss: 1.1355 - val_accuracy: 0.6099
Epoch 4/10
50000/50000 [==============================] - 8s 170us/sample - loss: 1.0133 - accuracy: 0.6543 -
val_loss: 1.1025 - val_accuracy: 0.6235
Epoch 5/10
50000/50000 [==============================] - 8s 166us/sample - loss: 0.9355 - accuracy: 0.6817 -
val_loss: 1.0970 - val_accuracy: 0.6254
Epoch 6/10
50000/50000 [==============================] - 8s 169us/sample - loss: 0.8727 - accuracy: 0.7043 -
val_loss: 1.0811 - val_accuracy: 0.6354
Epoch 7/10
50000/50000 [==============================] - 8s 163us/sample - loss: 0.8156 - accuracy: 0.7256 -
val_loss: 1.1796 - val_accuracy: 0.6199
Epoch 8/10
50000/50000 [==============================] - 8s 169us/sample - loss: 0.7610 - accuracy: 0.7443 -
val_loss: 1.1051 - val_accuracy: 0.6348
Epoch 9/10
50000/50000 [==============================] - 8s 165us/sample - loss: 0.7173 - accuracy: 0.7596 -
val_loss: 1.0848 - val_accuracy: 0.6505
```

ACTIVATION FUNCTION (RelU used):

```
class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes,learn_r,reg):
        super().__init__()

        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape, padding="same"))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='valid'))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='relu',activity_regularizer=regularizers.l2(reg)))
        self.add(Dense(84, activation='relu',activity_regularizer=regularizers.l2(reg)))
        #self.add(BatchNormalization())
        self.add(Dense(nb_classes, activation='softmax'))
        opt = Adam(lr=learn_r)
        self.compile(loss=categorical_crossentropy,metrics=['accuracy'], optimizer=opt)
```

BATCH NORMALIZATION:

```
# -*- coding: utf-8 -*-
"""
Created on Fri May  1 12:40:35 2020

@author: saikr
"""

from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.layers import Dense, Flatten, Conv2D, AveragePooling2D,BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adam
```

```
class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes,learn_r,reg):
        super().__init__()

        self.add(Conv2D(6, kernel_initializer='glorot_uniform', kernel_size=(5, 5), stride
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Conv2D(16, kernel_initializer='glorot_uniform', kernel_size=(5, 5), strid
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2
        self.add(Flatten())
        #self.add(Dense(120, activation='relu',activity_regulariz
        #self.add(Dense(84, activation='relu',activity_regularizer=regularizers.l2(reg)))
        self.add(Dense(120,kernel_initializer='glorot_uniform', activation='relu'))
        self.add(Dense(84, kernel_initializer='glorot_uniform',activation='relu'))
        '''self.add(Dense(120, activation='relu'))
        self.add(Dense(84, activation='relu'))'''
        self.add(BatchNormalization())
        self.add(Dense(nb_classes, activation='softmax'))
        opt = Adam(lr=learn_r)
        self.compile(loss=categorical_crossentropy,metrics=['accuracy'], optimizer=opt)
```

No documentation available

Click anywhere in this tool

Adding batch normalization, we get the following values:

```
Epoch 1/20
50000/50000 - 6s - loss: 1.4899 - accuracy: 0.4658 - val_loss: 1.3770 - val_accuracy: 0.5049
Epoch 2/20
50000/50000 - 6s - loss: 1.2112 - accuracy: 0.5720 - val_loss: 1.1729 - val_accuracy: 0.5869
Epoch 3/20
50000/50000 - 5s - loss: 1.0950 - accuracy: 0.6158 - val_loss: 1.2022 - val_accuracy: 0.5724
Epoch 4/20
50000/50000 - 5s - loss: 1.0193 - accuracy: 0.6423 - val_loss: 1.1731 - val_accuracy: 0.5856
Epoch 5/20
50000/50000 - 5s - loss: 0.9545 - accuracy: 0.6631 - val_loss: 1.0899 - val_accuracy: 0.6311
Epoch 6/20
50000/50000 - 5s - loss: 0.9055 - accuracy: 0.6819 - val_loss: 0.9696 - val_accuracy: 0.6626
Epoch 7/20
50000/50000 - 5s - loss: 0.8556 - accuracy: 0.6986 - val_loss: 0.9747 - val_accuracy: 0.6627
Epoch 8/20
50000/50000 - 5s - loss: 0.8135 - accuracy: 0.7147 - val_loss: 0.9556 - val_accuracy: 0.6729
```

LAYOUT OF THE SOLUTION:

a) Babysitting and training process:

The babysitting process consists of various checks and parameters that need to be performed and estimated:

i) Disabling regularization and checking for initial Loss

```
class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()

        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape, padding="same"))# tanh changed to relu
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='valid')) # tanh changed to relu
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='relu',activity_regularizer=regularizers.l2(0)))
        self.add(Dense(84, activation='relu',activity_regularizer=regularizers.l2(0)))
        self.add(Dense(nb_classes, activation='softmax'))

        self.compile(optimizer=opt,loss=categorical_crossentropy,metrics=['accuracy'])
```

```
In [6]: runfile('D:/SEM 2-spring 2020/COMPUTER VISION/project-3/CVTF.py', wdir='D:/SEM 2-spring 2020/
COMPUTER VISION/project-3')
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 9s 176us/sample - loss: 1.5176 - accuracy: 0.4550 -
val_loss: 1.3171 - val_accuracy: 0.5295
```

ii) Introducing regularization of (1e3) and observing an increase in the initial loss

```
        self.add(Dense(120, activation='relu',activity_regularizer=regularizers.l2(1e3)))
        self.add(Dense(84, activation='relu',activity_regularizer=regularizers.l2(1e3)))
        self.add(Dense(nb_classes, activation='softmax'))

        self.compile(optimizer=opt,loss=categorical_crossentropy,metrics=['accuracy'])
```

```
In [13]: runfile('D:/SEM 2-spring 2020/COMPUTER VISION/project-3/CVTF.py', wdir='D:/SEM 2-spring 2020/
COMPUTER VISION/project-3')
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 8s 167us/sample - loss: 2.7286 - accuracy: 0.0976 -
val_loss: 2.3027 - val_accuracy: 0.1000
```

Increase in loss from 1.5176 to 2.7286 is observed.

iii) Overfitting the model with only 20 data samples- and observing very small loss and training accuracy of 1-> with regularization set to 0

```
class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()

        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='relu', input_shape=input_shape, padding="same"))# tanh changed to relu
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding='valid')) # tanh changed to relu
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='relu',activity_regularizer=regularizers.l2(0)))
        self.add(Dense(84, activation='relu',activity_regularizer=regularizers.l2(0)))
        self.add(Dense(nb_classes, activation='softmax'))

        self.compile(optimizer=opt,loss=categorical_crossentropy,metrics=['accuracy'])
```

```
Epoch 48/50
20/20 [==============================] - 0s 997us/sample - loss: 0.1938 - accuracy: 1.0000 - val_loss: 3.5357 - val_accuracy:
0.1000
Epoch 49/50
20/20 [==============================] - 0s 898us/sample - loss: 0.1909 - accuracy: 1.0000 - val_loss: 3.5338 - val_accuracy:
0.1000
Epoch 50/50
20/20 [==============================] - 0s 947us/sample - loss: 0.1880 - accuracy: 1.0000 - val_loss: 3.5356 - val_accuracy:
0.1000
```

iv) Introducing small regularization and finding the learning rate that makes the loss go down:

a) Stage 1 : Low learning rate means Loss barely changes.
   Learning rate = 10**-8

```
#opt=Adam(learning_rate=lr)

opt=Adam(learning_rate=10**-8)

class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()

        self.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation=
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), paddi
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activatio
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), paddi
        self.add(Flatten())
        self.add(Dense(120, activation='relu',activity_regularizer=regula
        self.add(Dense(84, activation='relu',activity_regularizer=regulari
        self.add(Dense(nb_classes, activation='softmax'))

        self.compile(optimizer=opt,loss=categorical_crossentropy,metrics=
```

```
In [14]: runfile('D:/SEM 2-spring 2020/COMPUTER VISION/project-3/CVTF.py', wdir='D:/SEM 2-spring 2020/
COMPUTER VISION/project-3')
Reloaded modules: network
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 9s 173us/sample - loss: 2.4462 - accuracy: 0.1067 -
val_loss: 2.4398 - val_accuracy: 0.1056
Epoch 2/10
50000/50000 [==============================] - 8s 160us/sample - loss: 2.4444 - accuracy: 0.1067 -
val_loss: 2.4380 - val_accuracy: 0.1060
Epoch 3/10
50000/50000 [==============================] - 8s 166us/sample - loss: 2.4425 - accuracy: 0.1068 -
val_loss: 2.4362 - val_accuracy: 0.1065
```

Loss barely changes on keeping the learning rate too low

b) Stage 2: High learning rate means Loss becomes infinity or Nan. This is exploding loss. Learning rate is set to 10**8

```
#opt=Adam(learning_rate=lr)

opt=Adam(learning_rate=10**8)

class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()

        self.add(Conv2D(6, kernel_size=(5, 5), s
        self.add(AveragePooling2D(pool_size=(2,
        self.add(Conv2D(16, kernel_size=(5, 5),
        self.add(AveragePooling2D(pool_size=(2,
        self.add(Flatten())
        self.add(Dense(120, activation='relu',ac
        self.add(Dense(84, activation='relu',act
```

```
In [15]: runfile('D:/SEM 2-spring 2020/COMPUTER VISION/project-3/CVTF.py', wdir='D:/SEM 2-spring 2020/
COMPUTER VISION/project-3')
Reloaded modules: network
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 9s 184us/sample - loss: nan - accuracy: 0.1000 -
val_loss: nan - val_accuracy: 0.1000
Epoch 2/10
50000/50000 [==============================] - 8s 168us/sample - loss: nan - accuracy: 0.1000 -
val_loss: nan - val_accuracy: 0.1000
Epoch 3/10
50000/50000 [==============================] - 8s 166us/sample - loss: nan - accuracy: 0.1000 -
val_loss: nan - val_accuracy: 0.1000
Epoch 4/10
50000/50000 [==============================] - 8s 168us/sample - loss: nan - accuracy: 0.1000 -
val_loss: nan - val_accuracy: 0.1000
Epoch 5/10
50000/50000 [==============================] - 8s 166us/sample - loss: nan - accuracy: 0.1000 -
val_loss: nan - val_accuracy: 0.1000
```

We notice that because of the high learning rate, the loss becomes nan

v) Cross-validation strategy to find optimal values of regularization and learning rate.

a) Running coarse search-> using a larger interval and less number of epochs.

We use only 5 epochs for coarse tuning, and observe the following:

```
for count in range(2, 6):
    for iteration in range (0,10):
        iteration = iteration+1
        keras.backend.clear_session()
        #(xTrain,yTrain),(xTest,yTest) = datasets.cifar10.load_data()
        reg = random()*pow(10,np.random.randint(-10, 0))
        lr = random()*pow(10,-count)
        print("Learning rate: ",lr, "Regularisation", reg)
        model=LeNet(x_train[0].shape, num_classes, lr, reg)
        #print("[INFO] training model... ")
        model.fit(x=x_train, y=y_train, epochs=10, validation_data=(x_test, y_test), verbose=1)

H = model.fit(x_train, y_train,validation_data=(x_test, y_test),epochs=EPOCHS, verbose=2)
```

So, actually the range of learning rate values is from 10**-2 to 10**-6; and range of regularization is taken from 10** 0 to 10**-10

The following is how the results look :

b) Running fine search -> narrowing down the interval and using larger number of epochs.

For the fine search, we use more epochs (10) to narrow in on the optimal learning rate and regularization.

```
for count in range(2, 3):
    for iteration in range (0,10):
        iteration = iteration+1
        keras.backend.clear_session()
        #(xTrain,yTrain),(xTest,yTest) = datasets.cifar10.load_data()
        reg = random()*pow(10,np.random.randint(-5, 0))
        lr = random()*pow(10,-count)
        print("Learning rate: ",lr, "Regularisation", reg)
        model=LeNet(x_train[0].shape, num_classes, lr, reg)
        #print("[INFO] training model... ")
        model.fit(x=x_train, y=y_train, epochs=10, validation_data=(x_test, y_test), verbose=1)

H = model.fit(x_train, y_train,validation_data=(x_test, y_test),epochs=EPOCHS, verbose=2)
```

So, here the learning rate is taken from 10**-2 to 10**-3 and regularization is taken in the range of 10**0 to 10** -5

The output values look like the following:

```
In [43]: runfile('D:/SEM 2-spring 2020/COMPUTER VISION/project-3/CVTF.py', wdir='D:/SEM 2-spring 202
COMPUTER VISION/project-3')
Reloaded modules: network
learning rate:  0.004903134086896807 Regularisation 7.226318254107717e-06
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 11s 225us/sample - loss: 1.6276 - accuracy: 0.4086 -
val_loss: 1.4662 - val_accuracy: 0.4718
Epoch 2/10
50000/50000 [==============================] - 10s 206us/sample - loss: 1.3933 - accuracy: 0.4981 -
val_loss: 1.4500 - val_accuracy: 0.4774
Epoch 3/10
50000/50000 [==============================] - 11s 216us/sample - loss: 1.3055 - accuracy: 0.5371 -
val_loss: 1.3240 - val_accuracy: 0.5286
Epoch 4/10
50000/50000 [==============================] - 10s 205us/sample - loss: 1.2315 - accuracy: 0.5655 -
val_loss: 1.2387 - val_accuracy: 0.5686
Epoch 5/10
50000/50000 [==============================] - 10s 207us/sample - loss: 1.1780 - accuracy: 0.5873 -
val_loss: 1.2320 - val_accuracy: 0.5741
```

c) Stage 3: Finding the optimal learning rate that makes the loss go down- ideally by cross-validating in the range of [1e-3 to 1e-5]
The image below shows the best accuracy achieved at the end of10 epochs-which is 60.32.

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 11s 225us/sample - loss: 1.6276 - accuracy: 0.4086 -
val_loss: 1.4662 - val_accuracy: 0.4718
Epoch 2/10
50000/50000 [==============================] - 10s 206us/sample - loss: 1.3933 - accuracy: 0.4981 -
val_loss: 1.4500 - val_accuracy: 0.4774
Epoch 3/10
50000/50000 [==============================] - 11s 216us/sample - loss: 1.3055 - accuracy: 0.5371 -
val_loss: 1.3240 - val_accuracy: 0.5286
Epoch 4/10
50000/50000 [==============================] - 10s 205us/sample - loss: 1.2315 - accuracy: 0.5655 -
val_loss: 1.2387 - val_accuracy: 0.5686
Epoch 5/10
50000/50000 [==============================] - 10s 207us/sample - loss: 1.1780 - accuracy: 0.5873 -
val_loss: 1.2320 - val_accuracy: 0.5741
Epoch 6/10
50000/50000 [==============================] - 11s 217us/sample - loss: 1.1364 - accuracy: 0.6053 -
val_loss: 1.2247 - val_accuracy: 0.5834
Epoch 7/10
50000/50000 [==============================] - 12s 239us/sample - loss: 1.0897 - accuracy: 0.6221 -
val_loss: 1.1911 - val_accuracy: 0.5989
Epoch 8/10
50000/50000 [==============================] - 10s 202us/sample - loss: 1.0570 - accuracy: 0.6375 -
val_loss: 1.2464 - val_accuracy: 0.5896
Epoch 9/10
50000/50000 [==============================] - 10s 205us/sample - loss: 1.0339 - accuracy: 0.6467 -
val_loss: 1.1889 - val_accuracy: 0.5989
Epoch 10/10
50000/50000 [==============================] - 10s 208us/sample - loss: 1.0134 - accuracy: 0.6506 -
val_loss: 1.2192 - val_accuracy: 0.6032
```

So, The optimal learning rate and regularization obtained for this best value is :

```
learning rate:  0.0009403964457273863 Regularisation 5.790347893065 51e-06
```

TESTING THE NETWORK USING FACE AND NON-FACE DATA:

Now, we extend a similar LeNet 5 architecture to train the data on the FaceNonface dataset. We validate on 400 images. We have used 200 images for testing- 100 for face and 100 for non-face.

The babysitting follows the same approach, wherein the hyperparameters like regularization and learning rate are obtained using the coarse-fine cross validation strategy.

```
In [49]: runfile('D:/Neural Networks/Project_C/Code/facetrain.py', wdir='D:/Neural Networks/Project_C/
Code')
   ...: Reloaded modules: facenetwork
   ...: learning rate:  3.4517698687445954e-06 Regularisation 3.0449338243034084e-07
   ...: Train on 1600 samples, validate on 400 samples
   ...: Epoch 1/5
   ...:  - 1s - loss: 4.1262 - accuracy: 0.7163 - val_loss: 2.0089 - val_accuracy: 0.8650
   ...: Epoch 2/5
   ...:  - 1s - loss: 1.8567 - accuracy: 0.8794 - val_loss: 1.4717 - val_accuracy: 0.9000
   ...: Epoch 3/5
   ...:  - 1s - loss: 1.3269 - accuracy: 0.9187 - val_loss: 1.1475 - val_accuracy: 0.9150
   ...: Epoch 4/5
   ...:  - 1s - loss: 1.0101 - accuracy: 0.9375 - val_loss: 0.9213 - val_accuracy: 0.9325
   ...: Epoch 5/5
   ...:  - 1s - loss: 0.7977 - accuracy: 0.9450 - val_loss: 0.7698 - val_accuracy: 0.9450
   ...: learning rate:  1.7495317618275942e-07 Regularisation 6.81880070427967e-08
   ...: Train on 1600 samples, validate on 400 samples
   ...: Epoch 1/5
   ...:  - 1s - loss: 35.8310 - accuracy: 0.2750 - val_loss: 31.4070 - val_accuracy: 0.2375
   ...: Epoch 2/5
   ...:  - 1s - loss: 29.8107 - accuracy: 0.2569 - val_loss: 26.6859 - val_accuracy: 0.2475
   ...: Epoch 3/5
   ...:  - 1s - loss: 25.4746 - accuracy: 0.2562 - val_loss: 23.1960 - val_accuracy: 0.2875
   ...: Epoch 4/5
   ...:  - 1s - loss: 22.1022 - accuracy: 0.2812 - val_loss: 20.2605 - val_accuracy: 0.3250
   ...: Epoch 5/5
   ...:  - 1s - loss: 19.3416 - accuracy: 0.3212 - val_loss: 17.7757 - val_accuracy: 0.3550
   ...: learning rate:  4.8197909262343955e-06 Regularisation 9.3829366380755e-09
   ...: Train on 1600 samples, validate on 400 samples
   ...: Epoch 1/5
   ...:  - 1s - loss: 1.6478 - accuracy: 0.8319 - val_loss: 0.8702 - val_accuracy: 0.9075
   ...: Epoch 2/5
   ...:  - 1s - loss: 0.6080 - accuracy: 0.9175 - val_loss: 0.6245 - val_accuracy: 0.9275
   ...: Epoch 3/5
   ...:  - 1s - loss: 0.3310 - accuracy: 0.9406 - val_loss: 0.5313 - val_accuracy: 0.9450
   ...: Epoch 4/5
   ...:  - 1s - loss: 0.2053 - accuracy: 0.9650 - val_loss: 0.4868 - val_accuracy: 0.9475
   ...: Epoch 5/5
   ...:  - 1s - loss: 0.1520 - accuracy: 0.9775 - val_loss: 0.4599 - val_accuracy: 0.9500
   ...: learning rate:  7.132552766543269e-06 Regularisation 4.545792210116589e-07
   ...: Train on 1600 samples, validate on 400 samples
   ...: Epoch 1/5
```

Doing this fine cross validation strategy, we obtain the optimal parameters for learning rate and regularization which are **lr=2.3e-6 ; reg=9e-8.**

```
model = LeNet.build(128,128,3, num_classes,2.3e-6,9e-08)
opt = Adam(lr=2.3e-6)
model.compile(loss=categorical_crossentropy,metrics=['accuracy'], optimizer=opt)
EPOCHS = 20
H = model.fit(xTrain, yTrain,
              validation_data=(xVal, yVal),
            epochs=EPOCHS, verbose=1)
```

I have attached the .py files for training the model, the network used and the testing function.

The values for training look like:

```
Train on 1600 samples, validate on 400 samples
Epoch 1/20
1600/1600 [==============================] - 1s 716us/sample - loss: 0.7818 - accuracy:
0.5987 - val_loss: 0.7890 - val_accuracy: 0.6750
Epoch 2/20
1600/1600 [==============================] - 1s 318us/sample - loss: 0.6086 - accuracy:
0.7138 - val_loss: 0.5224 - val_accuracy: 0.7625
Epoch 3/20
1600/1600 [==============================] - 1s 317us/sample - loss: 0.5001 - accuracy:
0.7663 - val_loss: 0.4271 - val_accuracy: 0.8375
Epoch 4/20
1600/1600 [==============================] - 0s 307us/sample - loss: 0.4338 - accuracy:
0.8031 - val_loss: 0.3739 - val_accuracy: 0.8475
Epoch 5/20
1600/1600 [==============================] - 0s 299us/sample - loss: 0.3855 - accuracy:
0.8306 - val_loss: 0.3428 - val_accuracy: 0.8700
Epoch 6/20
1600/1600 [==============================] - 0s 312us/sample - loss: 0.3433 - accuracy:
0.8625 - val_loss: 0.3146 - val_accuracy: 0.8825
Epoch 7/20
1600/1600 [==============================] - 0s 309us/sample - loss: 0.3109 - accuracy:
0.8794 - val_loss: 0.2878 - val_accuracy: 0.9000
Epoch 8/20
1600/1600 [==============================] - 1s 327us/sample - loss: 0.3061 - accuracy:
0.8831 - val_loss: 0.2815 - val_accuracy: 0.9000
Epoch 9/20
1600/1600 [==============================] - 0s 312us/sample - loss: 0.2762 - accuracy:
0.8925 - val_loss: 0.2666 - val_accuracy: 0.9100
Epoch 10/20
1600/1600 [==============================] - 1s 331us/sample - loss: 0.2598 - accuracy:
```

```
Epoch 17/20
1600/1600 [==============================] - 1s 318us/sample - loss: 0.1863 - accuracy:
0.9419 - val_loss: 0.1826 - val_accuracy: 0.9400
Epoch 18/20
1600/1600 [==============================] - 1s 317us/sample - loss: 0.1707 - accuracy:
0.9500 - val_loss: 0.1582 - val_accuracy: 0.9525
Epoch 19/20
1600/1600 [==============================] - 1s 319us/sample - loss: 0.1651 - accuracy:
0.9500 - val_loss: 0.1636 - val_accuracy: 0.9525
Epoch 20/20
1600/1600 [==============================] - 1s 321us/sample - loss: 0.1588 - accuracy:
0.9513 - val_loss: 0.1483 - val_accuracy: 0.9650
WARNING:tensorflow:AutoGraph could not transform <function
canonicalize_signatures.<locals>.signature_wrapper at 0x000001CE9126D8B8> and will run
it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10
(on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
```

Final validation accuracy of 96.5% is obtained.

We follow a similar procedure for babysitting the network, and on testing we obtain the following output parameters on testing:

```
Reloaded modules: model
[INFO] loading network...
              precision    recall  f1-score   support

        Face       0.95      0.92      0.93       100
     NonFace       0.92      0.95      0.94       100

    accuracy                           0.94       200
   macro avg       0.94      0.94      0.93       200
weighted avg       0.94      0.94      0.93       200


In [67]:
```

ALL THE .PY FILES HAVE BEEN ATTACHED IN THE COMBINED ZIP FOLDER. PLEASE REFER TO THE readme.txt in the zip folder to know which file is for what.