HW: X-ray images classification

Before you begin, open Mobaxterm and connect to triton with the user and password you were give with. Activate the environment `2ndPaper` and then type the command `pip install scikit-image` .

In this assignment you will be dealing with classification of 32X32 X-ray images of the chest. The image can be classified into one of four options: lungs (l), clavicles (c), and heart (h) and background (b). Even though those labels are dependent, we will treat this task as multiclass and not as multilabel. The dataset for this assignment is located on a shared folder on triton (`/MLdata/MLcourse/X_ray/'`).

In [1]:
```
import os
import numpy as np
from tensorflow.keras.layers import Dense, MaxPool2D, Conv2D, Dropout
from tensorflow.keras.layers import Flatten, InputLayer
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import *

from tensorflow.keras.initializers import Constant
from tensorflow.keras.datasets import fashion_mnist
import tensorflow.keras.backend as K
from tensorflow.keras import regularizers
from tensorflow import keras
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import *
from skimage.io import imread

from skimage.transform import rescale, resize, downscale_local_mean
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="2"
```

In [2]:
```
import tensorflow as tf
config = tf.compat.v1.ConfigProto(gpu_options =
                         tf.compat.v1.GPUOptions(per_process_gpu_memory_fracti
on=0.8)
# device_count = {'GPU': 1}
)
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
tf.compat.v1.keras.backend.set_session(session)
```

In [3]:
```
def preprocess(datapath):
    # This part reads the images
    classes = ['b','c','l','h']
    imagelist = [fn for fn in os.listdir(datapath)]
    N = len(imagelist)
    num_classes = len(classes)
    images = np.zeros((N, 32, 32, 1))
    Y = np.zeros((N,num_classes))
    ii=0
    for fn in imagelist:

        src = imread(os.path.join(datapath, fn),1)
        img = resize(src,(32,32),order = 3)

        images[ii,:,:,0] = img
        cc = -1
        for cl in range(len(classes)):
            if fn[-5] == classes[cl]:
                cc = cl
        Y[ii,cc]=1
        ii += 1

    BaseImages = images
    BaseY = Y
    return BaseImages, BaseY
```

In [4]:
```
def preprocess_train_and_val(datapath):
    # This part reads the images
    classes = ['b','c','l','h']
    imagelist = [fn for fn in os.listdir(datapath)]
    N = len(imagelist)
    num_classes = len(classes)
    images = np.zeros((N, 32, 32, 1))
    Y = np.zeros((N,num_classes))
    ii=0
    for fn in imagelist:

        images[ii,:,:,0] = imread(os.path.join(datapath, fn),1)
        cc = -1
        for cl in range(len(classes)):
            if fn[-5] == classes[cl]:
                cc = cl
        Y[ii,cc]=1
        ii += 1

    return images, Y
```

In [5]:
```
#Loading the data for training and validation:
src_data = '/MLdata/MLcourse/X_ray/'
train_path = src_data + 'train'
val_path = src_data + 'validation'
test_path = src_data + 'test'
BaseX_train , BaseY_train = preprocess_train_and_val(train_path)
BaseX_val , BaseY_val = preprocess_train_and_val(val_path)
X_test, Y_test = preprocess(test_path)
```

In [6]:
```
keras.backend.clear_session()
```

PART 1: Fully connected layers

Task 1: *NN with fully connected layers.

Elaborate a NN with 2 hidden fully connected layers with 300, 150 neurons and 4 neurons for classification. Use ReLU activation functions for the hidden layers and He_normal for initialization. Don't forget to flatten your image before feedforward to the first dense layer. Name the model `model_relu` .*

In [7]:
```
#--------------------------Impelment your code here:-------------------------------------
from tensorflow.keras.initializers import he_normal

model_relu = Sequential(name="model_relu")
model_relu.add(Flatten(input_shape=(32, 32, 1)))
model_relu.add(Dense(300, activation='relu', kernel_initializer=tf.keras.initializers.he_normal()))
model_relu.add(Dense(150))
model_relu.add(Activation('relu'))
model_relu.add(Dense(4))
model_relu.add(Activation('softmax'))
#-------------------------------------------------------------------------------------
```

In [8]:
```
model_relu.summary()
```

```
Model: "model_relu"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 300)               307500    
_________________________________________________________________
dense_1 (Dense)              (None, 150)               45150     
_________________________________________________________________
activation (Activation)      (None, 150)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 604       
_________________________________________________________________
activation_1 (Activation)    (None, 4)                 0         
=================================================================
Total params: 353,254
Trainable params: 353,254
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
#Inputs:
input_shape = (32,32,1)
learn_rate = 1e-5
decay = 0
batch_size = 64
epochs = 25

#Define your optimizar parameters:
AdamOpt = Adam(lr=learn_rate,decay=decay)
```

Compile the model with the optimizer above, accuracy metric and adequate loss for multiclass task. Train your model on the training set and evaluate the model on the testing set. Print the accuracy and loss over the testing set.

In [10]:
```
#--------------------------Impelment your code here:-------------------------------------
model_relu.compile(optimizer=AdamOpt, metrics=['accuracy'], loss='categorical_crossentropy')
history_relu = model_relu.fit(BaseX_train, BaseY_train, batch_size=batch_size, epochs=epochs,verbose=2, validation_data=(BaseX_val, BaseY_val))
results_metrics = model_relu.evaluate(X_test, Y_test, batch_size=batch_size)
print("Test loss, Test accuracy:", results_metrics)
#--------------------------------------------------------------------------------------
```

```
Epoch 1/25
102/102 - 1s - loss: 1.3252 - accuracy: 0.3630 - val_loss: 1.2408 - val_accur
acy: 0.5301
Epoch 2/25
102/102 - 0s - loss: 1.1845 - accuracy: 0.5748 - val_loss: 1.1275 - val_accur
acy: 0.6065
Epoch 3/25
102/102 - 0s - loss: 1.0932 - accuracy: 0.6345 - val_loss: 1.0524 - val_accur
acy: 0.6499
Epoch 4/25
102/102 - 0s - loss: 1.0270 - accuracy: 0.6660 - val_loss: 0.9963 - val_accur
acy: 0.6881
Epoch 5/25
102/102 - 0s - loss: 0.9743 - accuracy: 0.6960 - val_loss: 0.9501 - val_accur
acy: 0.7037
Epoch 6/25
102/102 - 0s - loss: 0.9317 - accuracy: 0.7102 - val_loss: 0.9134 - val_accur
acy: 0.7240
Epoch 7/25
102/102 - 0s - loss: 0.8947 - accuracy: 0.7263 - val_loss: 0.8793 - val_accur
acy: 0.7297
Epoch 8/25
102/102 - 0s - loss: 0.8604 - accuracy: 0.7374 - val_loss: 0.8507 - val_accur
acy: 0.7373
Epoch 9/25
102/102 - 0s - loss: 0.8306 - accuracy: 0.7481 - val_loss: 0.8225 - val_accur
acy: 0.7483
Epoch 10/25
102/102 - 0s - loss: 0.8032 - accuracy: 0.7576 - val_loss: 0.7987 - val_accur
acy: 0.7569
Epoch 11/25
102/102 - 0s - loss: 0.7794 - accuracy: 0.7695 - val_loss: 0.7771 - val_accur
acy: 0.7616
Epoch 12/25
102/102 - 0s - loss: 0.7567 - accuracy: 0.7723 - val_loss: 0.7570 - val_accur
acy: 0.7668
Epoch 13/25
102/102 - 0s - loss: 0.7373 - accuracy: 0.7783 - val_loss: 0.7378 - val_accur
acy: 0.7726
Epoch 14/25
102/102 - 0s - loss: 0.7181 - accuracy: 0.7830 - val_loss: 0.7211 - val_accur
acy: 0.7824
Epoch 15/25
102/102 - 0s - loss: 0.7001 - accuracy: 0.7892 - val_loss: 0.7050 - val_accur
acy: 0.7841
Epoch 16/25
102/102 - 0s - loss: 0.6832 - accuracy: 0.7946 - val_loss: 0.6895 - val_accur
acy: 0.7882
Epoch 17/25
102/102 - 0s - loss: 0.6675 - accuracy: 0.7978 - val_loss: 0.6763 - val_accur
acy: 0.7911
Epoch 18/25
102/102 - 0s - loss: 0.6522 - accuracy: 0.8032 - val_loss: 0.6652 - val_accur
acy: 0.7894
Epoch 19/25
102/102 - 0s - loss: 0.6387 - accuracy: 0.8068 - val_loss: 0.6471 - val_accur
acy: 0.7969
```

```
Epoch 20/25
102/102 - 0s - loss: 0.6244 - accuracy: 0.8099 - val_loss: 0.6362 - val_accur
acy: 0.8009
Epoch 21/25
102/102 - 0s - loss: 0.6123 - accuracy: 0.8125 - val_loss: 0.6238 - val_accur
acy: 0.8032
Epoch 22/25
102/102 - 0s - loss: 0.5999 - accuracy: 0.8177 - val_loss: 0.6131 - val_accur
acy: 0.8084
Epoch 23/25
102/102 - 0s - loss: 0.5886 - accuracy: 0.8193 - val_loss: 0.6059 - val_accur
acy: 0.8061
Epoch 24/25
102/102 - 0s - loss: 0.5773 - accuracy: 0.8253 - val_loss: 0.5909 - val_accur
acy: 0.8113
Epoch 25/25
102/102 - 0s - loss: 0.5660 - accuracy: 0.8272 - val_loss: 0.5827 - val_accur
acy: 0.8160
3/3 [==============================] - 0s 4ms/step - loss: 0.7957 - accuracy:
0.6800
Test loss, Test accuracy: [0.7957050204277039, 0.6800000071525574]
```

Task 2: *Activation functions.*

Change the activation functions to LeakyRelu or tanh or sigmoid. Name the new model `new_a_model` . Explain how it can affect the model.*

In [11]:
```
#--------------------------Impelment your code here:-------------------------
-----------
new_a_model = Sequential(name="new_a_model")
new_a_model.add(Flatten(input_shape=(32,32,1)))
new_a_model.add(Dense(300,activation='sigmoid', kernel_initializer=tf.keras.in
itializers.he_normal()))
new_a_model.add(Dense(150))
new_a_model.add(Activation('tanh'))
new_a_model.add(Dense(4))
new_a_model.add(Activation('softmax'))
#----------------------------------------------------------------------------
-----------
```

In [12]:
```
new_a_model.summary()
```

```
Model: "new_a_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 1024)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 300)               307500    
_________________________________________________________________
dense_4 (Dense)              (None, 150)               45150     
_________________________________________________________________
activation_2 (Activation)    (None, 150)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 604       
_________________________________________________________________
activation_3 (Activation)    (None, 4)                 0         
=================================================================
Total params: 353,254
Trainable params: 353,254
Non-trainable params: 0
_________________________________________________________________
```

Task 3: *Number of epochs.*

Train the new model using 25 and 40 epochs. What difference does it makes in term of performance? Remember to save the compiled model for having initialized weights for every run as we did in tutorial 12. Evaluate each trained model on the test set*

In [13]:
```
#Inputs:
input_shape = (32,32,1)
learn_rate = 1e-5
decay = 0
batch_size = 64
epochs = 25

#Defining the optimizar parameters:
AdamOpt = Adam(lr=learn_rate,decay=decay)
```

In [14]:
```
#--------------------------Impelment your code here:---------------------------------------
new_a_model.compile(optimizer=AdamOpt, metrics=['accuracy'], loss='categorical_crossentropy')

# saving weights:
if not("init_weights" in os.listdir()):
    os.mkdir("init_weights")
save_dir = "init_weights/"
model_name = "initial_weights"
model_path = os.path.join(save_dir, model_name)
new_a_model.save(model_path)

history_new_a = new_a_model.fit(BaseX_train, BaseY_train, batch_size=batch_size, epochs=epochs, verbose=2, validation_data=(BaseX_val, BaseY_val))
results_metrics = new_a_model.evaluate(X_test, Y_test, batch_size=batch_size)
print("Test loss, Test accuracy:", results_metrics)
#-------------------------------------------------------------------------------------
```

```
INFO:tensorflow:Assets written to: init_weights/initial_weights/assets
Epoch 1/25
102/102 - 1s - loss: 1.3973 - accuracy: 0.3267 - val_loss: 1.3272 - val_accur
acy: 0.3918
Epoch 2/25
102/102 - 0s - loss: 1.2963 - accuracy: 0.4282 - val_loss: 1.2640 - val_accur
acy: 0.5191
Epoch 3/25
102/102 - 0s - loss: 1.2394 - accuracy: 0.5388 - val_loss: 1.2094 - val_accur
acy: 0.5579
Epoch 4/25
102/102 - 0s - loss: 1.1906 - accuracy: 0.5544 - val_loss: 1.1628 - val_accur
acy: 0.5660
Epoch 5/25
102/102 - 0s - loss: 1.1500 - accuracy: 0.5635 - val_loss: 1.1249 - val_accur
acy: 0.5741
Epoch 6/25
102/102 - 0s - loss: 1.1162 - accuracy: 0.5731 - val_loss: 1.0931 - val_accur
acy: 0.5845
Epoch 7/25
102/102 - 0s - loss: 1.0875 - accuracy: 0.5817 - val_loss: 1.0659 - val_accur
acy: 0.6030
Epoch 8/25
102/102 - 0s - loss: 1.0633 - accuracy: 0.5965 - val_loss: 1.0440 - val_accur
acy: 0.6123
Epoch 9/25
102/102 - 0s - loss: 1.0429 - accuracy: 0.5944 - val_loss: 1.0240 - val_accur
acy: 0.6233
Epoch 10/25
102/102 - 0s - loss: 1.0245 - accuracy: 0.6115 - val_loss: 1.0075 - val_accur
acy: 0.6186
Epoch 11/25
102/102 - 0s - loss: 1.0081 - accuracy: 0.6155 - val_loss: 0.9936 - val_accur
acy: 0.6233
Epoch 12/25
102/102 - 0s - loss: 0.9941 - accuracy: 0.6196 - val_loss: 0.9800 - val_accur
acy: 0.6285
Epoch 13/25
102/102 - 0s - loss: 0.9812 - accuracy: 0.6231 - val_loss: 0.9675 - val_accur
acy: 0.6429
Epoch 14/25
102/102 - 0s - loss: 0.9698 - accuracy: 0.6299 - val_loss: 0.9571 - val_accur
acy: 0.6470
Epoch 15/25
102/102 - 0s - loss: 0.9581 - accuracy: 0.6328 - val_loss: 0.9482 - val_accur
acy: 0.6458
Epoch 16/25
102/102 - 0s - loss: 0.9479 - accuracy: 0.6386 - val_loss: 0.9373 - val_accur
acy: 0.6557
Epoch 17/25
102/102 - 0s - loss: 0.9375 - accuracy: 0.6433 - val_loss: 0.9283 - val_accur
acy: 0.6632
Epoch 18/25
102/102 - 0s - loss: 0.9279 - accuracy: 0.6483 - val_loss: 0.9195 - val_accur
acy: 0.6620
Epoch 19/25
102/102 - 0s - loss: 0.9196 - accuracy: 0.6492 - val_loss: 0.9129 - val_accur
```

```
acy: 0.6586
Epoch 20/25
102/102 - 0s - loss: 0.9108 - accuracy: 0.6546 - val_loss: 0.9048 - val_accur
acy: 0.6661
Epoch 21/25
102/102 - 0s - loss: 0.9029 - accuracy: 0.6594 - val_loss: 0.8992 - val_accur
acy: 0.6696
Epoch 22/25
102/102 - 0s - loss: 0.8947 - accuracy: 0.6627 - val_loss: 0.8896 - val_accur
acy: 0.6696
Epoch 23/25
102/102 - 0s - loss: 0.8868 - accuracy: 0.6674 - val_loss: 0.8851 - val_accur
acy: 0.6794
Epoch 24/25
102/102 - 0s - loss: 0.8804 - accuracy: 0.6718 - val_loss: 0.8762 - val_accur
acy: 0.6800
Epoch 25/25
102/102 - 0s - loss: 0.8724 - accuracy: 0.6735 - val_loss: 0.8702 - val_accur
acy: 0.6823
3/3 [==============================] - 0s 4ms/step - loss: 0.9879 - accuracy:
0.5714
Test loss, Test accuracy: [0.9878899455070496, 0.5714285969734192]
```

In [15]:

```
#Inputs:
input_shape = (32,32,1)
learn_rate = 1e-5
decay = 0
batch_size = 64
epochs = 40

#Defining the optimizar parameters:
AdamOpt = Adam(lr=learn_rate,decay=decay)
```

In [16]:
```
#--------------------------Impelment your code here:---------------------------
-----------
from tensorflow.keras.models import load_model

new_a_model=load_model("init_weights/initial_weights")

history_new_a = new_a_model.fit(BaseX_train, BaseY_train, batch_size=batch_siz
e, epochs=epochs, verbose=2, validation_data=(BaseX_val, BaseY_val))
results_metrics = new_a_model.evaluate(X_test, Y_test, batch_size=batch_size)
print("Test loss, Test accuracy:", results_metrics)
#-------------------------------------------------------------------------------
------------
```

```
Epoch 1/40
102/102 - 1s - loss: 1.3987 - accuracy: 0.3315 - val_loss: 1.3277 - val_accur
acy: 0.3918
Epoch 2/40
102/102 - 0s - loss: 1.2967 - accuracy: 0.4416 - val_loss: 1.2642 - val_accur
acy: 0.5035
Epoch 3/40
102/102 - 0s - loss: 1.2399 - accuracy: 0.5185 - val_loss: 1.2105 - val_accur
acy: 0.5579
Epoch 4/40
102/102 - 0s - loss: 1.1915 - accuracy: 0.5609 - val_loss: 1.1662 - val_accur
acy: 0.5677
Epoch 5/40
102/102 - 0s - loss: 1.1517 - accuracy: 0.5615 - val_loss: 1.1269 - val_accur
acy: 0.5741
Epoch 6/40
102/102 - 0s - loss: 1.1173 - accuracy: 0.5721 - val_loss: 1.0937 - val_accur
acy: 0.5880
Epoch 7/40
102/102 - 0s - loss: 1.0884 - accuracy: 0.5845 - val_loss: 1.0668 - val_accur
acy: 0.5903
Epoch 8/40
102/102 - 0s - loss: 1.0635 - accuracy: 0.5891 - val_loss: 1.0446 - val_accur
acy: 0.5972
Epoch 9/40
102/102 - 0s - loss: 1.0435 - accuracy: 0.5951 - val_loss: 1.0244 - val_accur
acy: 0.6244
Epoch 10/40
102/102 - 0s - loss: 1.0251 - accuracy: 0.6112 - val_loss: 1.0082 - val_accur
acy: 0.6221
Epoch 11/40
102/102 - 0s - loss: 1.0087 - accuracy: 0.6137 - val_loss: 0.9931 - val_accur
acy: 0.6296
Epoch 12/40
102/102 - 0s - loss: 0.9945 - accuracy: 0.6222 - val_loss: 0.9796 - val_accur
acy: 0.6360
Epoch 13/40
102/102 - 0s - loss: 0.9817 - accuracy: 0.6247 - val_loss: 0.9682 - val_accur
acy: 0.6337
Epoch 14/40
102/102 - 0s - loss: 0.9691 - accuracy: 0.6282 - val_loss: 0.9571 - val_accur
acy: 0.6505
Epoch 15/40
102/102 - 0s - loss: 0.9591 - accuracy: 0.6331 - val_loss: 0.9492 - val_accur
acy: 0.6429
Epoch 16/40
102/102 - 0s - loss: 0.9485 - accuracy: 0.6389 - val_loss: 0.9374 - val_accur
acy: 0.6562
Epoch 17/40
102/102 - 0s - loss: 0.9381 - accuracy: 0.6424 - val_loss: 0.9287 - val_accur
acy: 0.6551
Epoch 18/40
102/102 - 0s - loss: 0.9288 - accuracy: 0.6463 - val_loss: 0.9206 - val_accur
acy: 0.6632
Epoch 19/40
102/102 - 0s - loss: 0.9207 - accuracy: 0.6508 - val_loss: 0.9133 - val_accur
acy: 0.6528
```

```
Epoch 20/40
102/102 - 0s - loss: 0.9120 - accuracy: 0.6557 - val_loss: 0.9048 - val_accur
acy: 0.6644
Epoch 21/40
102/102 - 0s - loss: 0.9032 - accuracy: 0.6603 - val_loss: 0.8997 - val_accur
acy: 0.6678
Epoch 22/40
102/102 - 0s - loss: 0.8955 - accuracy: 0.6637 - val_loss: 0.8918 - val_accur
acy: 0.6713
Epoch 23/40
102/102 - 0s - loss: 0.8888 - accuracy: 0.6671 - val_loss: 0.8839 - val_accur
acy: 0.6742
Epoch 24/40
102/102 - 0s - loss: 0.8815 - accuracy: 0.6708 - val_loss: 0.8779 - val_accur
acy: 0.6742
Epoch 25/40
102/102 - 0s - loss: 0.8742 - accuracy: 0.6708 - val_loss: 0.8722 - val_accur
acy: 0.6817
Epoch 26/40
102/102 - 0s - loss: 0.8678 - accuracy: 0.6783 - val_loss: 0.8658 - val_accur
acy: 0.6817
Epoch 27/40
102/102 - 0s - loss: 0.8612 - accuracy: 0.6784 - val_loss: 0.8592 - val_accur
acy: 0.6846
Epoch 28/40
102/102 - 0s - loss: 0.8541 - accuracy: 0.6852 - val_loss: 0.8542 - val_accur
acy: 0.6840
Epoch 29/40
102/102 - 0s - loss: 0.8479 - accuracy: 0.6864 - val_loss: 0.8478 - val_accur
acy: 0.6921
Epoch 30/40
102/102 - 0s - loss: 0.8415 - accuracy: 0.6892 - val_loss: 0.8429 - val_accur
acy: 0.6916
Epoch 31/40
102/102 - 0s - loss: 0.8355 - accuracy: 0.6894 - val_loss: 0.8369 - val_accur
acy: 0.6956
Epoch 32/40
102/102 - 0s - loss: 0.8304 - accuracy: 0.6922 - val_loss: 0.8319 - val_accur
acy: 0.7002
Epoch 33/40
102/102 - 0s - loss: 0.8236 - accuracy: 0.6943 - val_loss: 0.8295 - val_accur
acy: 0.7014
Epoch 34/40
102/102 - 0s - loss: 0.8182 - accuracy: 0.6954 - val_loss: 0.8227 - val_accur
acy: 0.7025
Epoch 35/40
102/102 - 0s - loss: 0.8130 - accuracy: 0.6969 - val_loss: 0.8182 - val_accur
acy: 0.7054
Epoch 36/40
102/102 - 0s - loss: 0.8082 - accuracy: 0.6996 - val_loss: 0.8153 - val_accur
acy: 0.7031
Epoch 37/40
102/102 - 0s - loss: 0.8025 - accuracy: 0.6996 - val_loss: 0.8083 - val_accur
acy: 0.7049
Epoch 38/40
102/102 - 0s - loss: 0.7981 - accuracy: 0.7030 - val_loss: 0.8051 - val_accur
acy: 0.7072
```

```
Epoch 39/40
102/102 - 0s - loss: 0.7931 - accuracy: 0.7051 - val_loss: 0.8000 - val_accur
acy: 0.7072
Epoch 40/40
102/102 - 0s - loss: 0.7886 - accuracy: 0.7059 - val_loss: 0.7958 - val_accur
acy: 0.7089
3/3 [==============================] - 0s 4ms/step - loss: 0.9317 - accuracy:
0.5943
Test loss, Test accuracy: [0.9316574335098267, 0.5942857265472412]
```

Task 4: *Mini-batches.*

Build the `model_relu` again and run it with a batch size of 32 instead of 64. What are the advantages of the mini-batch vs. SGD?*

In [17]:
```
keras.backend.clear_session()
```

In [18]:
```
#--------------------------Impelment your code here:-------------------------
-----------
model_relu = Sequential(name="model_relu")
model_relu.add(Flatten(input_shape=(32, 32, 1)))
model_relu.add(Dense(300,activation='relu', kernel_initializer=tf.keras.initia
lizers.he_normal()))
model_relu.add(Dense(150))
model_relu.add(Activation('relu'))
model_relu.add(Dense(4))
model_relu.add(Activation('softmax'))
#----------------------------------------------------------------------------
-----------
```

In [19]:
```
batch_size = 32
epochs = 50

#Define your optimizar parameters:
AdamOpt = Adam(lr=learn_rate,decay=decay)
```

In [20]:
```
#--------------------------Impelment your code here:---------------------------
-----------
model_relu.compile(optimizer=AdamOpt,metrics=['accuracy'], loss='categorical_c
rossentropy')
history_relu = model_relu.fit(BaseX_train, BaseY_train, batch_size=batch_size,
epochs=epochs, verbose=2, validation_data=(BaseX_val, BaseY_val))
y_pred_test = model_relu.predict(X_test)
results_metrics = model_relu.evaluate(X_test, Y_test, batch_size=batch_size)
print("Test loss, Test accuracy:", results_metrics)
#------------------------------------------------------------------------------
-----------
```

```
Epoch 1/50
203/203 - 1s - loss: 1.2279 - accuracy: 0.5073 - val_loss: 1.1124 - val_accur
acy: 0.6047
Epoch 2/50
203/203 - 1s - loss: 1.0546 - accuracy: 0.6452 - val_loss: 1.0010 - val_accur
acy: 0.6898
Epoch 3/50
203/203 - 1s - loss: 0.9674 - accuracy: 0.6954 - val_loss: 0.9309 - val_accur
acy: 0.7078
Epoch 4/50
203/203 - 1s - loss: 0.9039 - accuracy: 0.7156 - val_loss: 0.8779 - val_accur
acy: 0.7269
Epoch 5/50
203/203 - 1s - loss: 0.8524 - accuracy: 0.7349 - val_loss: 0.8340 - val_accur
acy: 0.7407
Epoch 6/50
203/203 - 1s - loss: 0.8093 - accuracy: 0.7504 - val_loss: 0.7973 - val_accur
acy: 0.7506
Epoch 7/50
203/203 - 1s - loss: 0.7739 - accuracy: 0.7615 - val_loss: 0.7654 - val_accur
acy: 0.7604
Epoch 8/50
203/203 - 1s - loss: 0.7418 - accuracy: 0.7734 - val_loss: 0.7391 - val_accur
acy: 0.7691
Epoch 9/50
203/203 - 1s - loss: 0.7126 - accuracy: 0.7804 - val_loss: 0.7100 - val_accur
acy: 0.7801
Epoch 10/50
203/203 - 1s - loss: 0.6857 - accuracy: 0.7901 - val_loss: 0.6836 - val_accur
acy: 0.7818
Epoch 11/50
203/203 - 1s - loss: 0.6610 - accuracy: 0.7980 - val_loss: 0.6642 - val_accur
acy: 0.7928
Epoch 12/50
203/203 - 1s - loss: 0.6382 - accuracy: 0.8061 - val_loss: 0.6460 - val_accur
acy: 0.7940
Epoch 13/50
203/203 - 1s - loss: 0.6191 - accuracy: 0.8105 - val_loss: 0.6258 - val_accur
acy: 0.8015
Epoch 14/50
203/203 - 1s - loss: 0.6003 - accuracy: 0.8136 - val_loss: 0.6090 - val_accur
acy: 0.8073
Epoch 15/50
203/203 - 1s - loss: 0.5827 - accuracy: 0.8182 - val_loss: 0.5926 - val_accur
acy: 0.8131
Epoch 16/50
203/203 - 1s - loss: 0.5673 - accuracy: 0.8258 - val_loss: 0.5773 - val_accur
acy: 0.8177
Epoch 17/50
203/203 - 1s - loss: 0.5515 - accuracy: 0.8323 - val_loss: 0.5637 - val_accur
acy: 0.8235
Epoch 18/50
203/203 - 1s - loss: 0.5380 - accuracy: 0.8341 - val_loss: 0.5523 - val_accur
acy: 0.8281
Epoch 19/50
203/203 - 1s - loss: 0.5246 - accuracy: 0.8400 - val_loss: 0.5389 - val_accur
acy: 0.8310
```

```
Epoch 20/50
203/203 - 1s - loss: 0.5117 - accuracy: 0.8407 - val_loss: 0.5256 - val_accur
acy: 0.8368
Epoch 21/50
203/203 - 1s - loss: 0.5001 - accuracy: 0.8434 - val_loss: 0.5174 - val_accur
acy: 0.8368
Epoch 22/50
203/203 - 1s - loss: 0.4893 - accuracy: 0.8506 - val_loss: 0.5098 - val_accur
acy: 0.8397
Epoch 23/50
203/203 - 1s - loss: 0.4800 - accuracy: 0.8513 - val_loss: 0.4983 - val_accur
acy: 0.8438
Epoch 24/50
203/203 - 1s - loss: 0.4699 - accuracy: 0.8534 - val_loss: 0.4873 - val_accur
acy: 0.8426
Epoch 25/50
203/203 - 1s - loss: 0.4600 - accuracy: 0.8567 - val_loss: 0.4787 - val_accur
acy: 0.8461
Epoch 26/50
203/203 - 1s - loss: 0.4518 - accuracy: 0.8590 - val_loss: 0.4695 - val_accur
acy: 0.8490
Epoch 27/50
203/203 - 1s - loss: 0.4433 - accuracy: 0.8607 - val_loss: 0.4648 - val_accur
acy: 0.8547
Epoch 28/50
203/203 - 1s - loss: 0.4349 - accuracy: 0.8639 - val_loss: 0.4633 - val_accur
acy: 0.8530
Epoch 29/50
203/203 - 1s - loss: 0.4287 - accuracy: 0.8669 - val_loss: 0.4502 - val_accur
acy: 0.8565
Epoch 30/50
203/203 - 1s - loss: 0.4222 - accuracy: 0.8653 - val_loss: 0.4450 - val_accur
acy: 0.8582
Epoch 31/50
203/203 - 1s - loss: 0.4143 - accuracy: 0.8706 - val_loss: 0.4416 - val_accur
acy: 0.8623
Epoch 32/50
203/203 - 1s - loss: 0.4081 - accuracy: 0.8723 - val_loss: 0.4308 - val_accur
acy: 0.8611
Epoch 33/50
203/203 - 1s - loss: 0.4013 - accuracy: 0.8715 - val_loss: 0.4268 - val_accur
acy: 0.8634
Epoch 34/50
203/203 - 1s - loss: 0.3958 - accuracy: 0.8755 - val_loss: 0.4239 - val_accur
acy: 0.8652
Epoch 35/50
203/203 - 1s - loss: 0.3902 - accuracy: 0.8772 - val_loss: 0.4193 - val_accur
acy: 0.8709
Epoch 36/50
203/203 - 1s - loss: 0.3843 - accuracy: 0.8786 - val_loss: 0.4121 - val_accur
acy: 0.8675
Epoch 37/50
203/203 - 1s - loss: 0.3798 - accuracy: 0.8806 - val_loss: 0.4085 - val_accur
acy: 0.8721
Epoch 38/50
203/203 - 1s - loss: 0.3742 - accuracy: 0.8814 - val_loss: 0.4051 - val_accur
acy: 0.8704
```

```
Epoch 39/50
203/203 - 1s - loss: 0.3700 - accuracy: 0.8832 - val_loss: 0.3997 - val_accur
acy: 0.8727
Epoch 40/50
203/203 - 1s - loss: 0.3655 - accuracy: 0.8837 - val_loss: 0.3966 - val_accur
acy: 0.8744
Epoch 41/50
203/203 - 1s - loss: 0.3599 - accuracy: 0.8866 - val_loss: 0.3934 - val_accur
acy: 0.8779
Epoch 42/50
203/203 - 1s - loss: 0.3567 - accuracy: 0.8866 - val_loss: 0.3904 - val_accur
acy: 0.8779
Epoch 43/50
203/203 - 1s - loss: 0.3521 - accuracy: 0.8877 - val_loss: 0.3869 - val_accur
acy: 0.8802
Epoch 44/50
203/203 - 1s - loss: 0.3492 - accuracy: 0.8877 - val_loss: 0.3825 - val_accur
acy: 0.8808
Epoch 45/50
203/203 - 1s - loss: 0.3438 - accuracy: 0.8913 - val_loss: 0.3805 - val_accur
acy: 0.8819
Epoch 46/50
203/203 - 1s - loss: 0.3411 - accuracy: 0.8926 - val_loss: 0.3752 - val_accur
acy: 0.8791
Epoch 47/50
203/203 - 1s - loss: 0.3367 - accuracy: 0.8930 - val_loss: 0.3737 - val_accur
acy: 0.8819
Epoch 48/50
203/203 - 1s - loss: 0.3337 - accuracy: 0.8928 - val_loss: 0.3694 - val_accur
acy: 0.8819
Epoch 49/50
203/203 - 1s - loss: 0.3303 - accuracy: 0.8940 - val_loss: 0.3642 - val_accur
acy: 0.8825
Epoch 50/50
203/203 - 1s - loss: 0.3271 - accuracy: 0.8950 - val_loss: 0.3640 - val_accur
acy: 0.8883
6/6 [==============================] - 0s 3ms/step - loss: 0.8569 - accuracy:
0.6629
Test loss, Test accuracy: [0.8569125533103943, 0.6628571152687073]
```

Task 4: *Batch normalization.*

Build the `new_a_model` again and add batch normalization layers. How does it impact your results?*

In [21]:

```
keras.backend.clear_session()
```

In [22]:
```
#--------------------------Impelment your code here:--------------------------
-----------
new_a_model = Sequential(name="new_a_model")
new_a_model.add(Flatten(input_shape=(32,32,1)))
new_a_model.add(Dense(300,activation='sigmoid', kernel_initializer=tf.keras.in
itializers.he_normal()))
new_a_model.add(BatchNormalization())
new_a_model.add(Dense(150))
new_a_model.add(BatchNormalization())
new_a_model.add(Activation('tanh'))
new_a_model.add(Dense(4))
new_a_model.add(BatchNormalization())
new_a_model.add(Activation('softmax'))
#----------------------------------------------------------------------------
----------
```

In [23]:
```
batch_size = 32
epochs = 50

#Define your optimizar parameters:
AdamOpt = Adam(lr=learn_rate,decay=decay)
#Compile the network:
```

In [24]:
```
#Preforming the training by using fit
#--------------------------Impelment your code here:-------------------------------------
new_a_model.compile(optimizer=AdamOpt,metrics=['accuracy'], loss='categorical_crossentropy')
history_new_a = new_a_model.fit(BaseX_train, BaseY_train, batch_size=batch_size, epochs=epochs, verbose=2, validation_data=(BaseX_val, BaseY_val))
y_pred_test = new_a_model.predict(X_test)
results_metrics = new_a_model.evaluate(X_test, Y_test, batch_size=batch_size)
print("Test loss, Test accuracy:", results_metrics)
#-----------------------------------------------------------------------------------------
```

```
Epoch 1/50
203/203 - 2s - loss: 1.1146 - accuracy: 0.5431 - val_loss: 1.2678 - val_accur
acy: 0.3171
Epoch 2/50
203/203 - 1s - loss: 0.8384 - accuracy: 0.7047 - val_loss: 0.9787 - val_accur
acy: 0.6713
Epoch 3/50
203/203 - 1s - loss: 0.7532 - accuracy: 0.7569 - val_loss: 0.7656 - val_accur
acy: 0.7691
Epoch 4/50
203/203 - 1s - loss: 0.7073 - accuracy: 0.7830 - val_loss: 0.6745 - val_accur
acy: 0.8194
Epoch 5/50
203/203 - 1s - loss: 0.6782 - accuracy: 0.8001 - val_loss: 0.6482 - val_accur
acy: 0.8171
Epoch 6/50
203/203 - 1s - loss: 0.6440 - accuracy: 0.8202 - val_loss: 0.6368 - val_accur
acy: 0.8322
Epoch 7/50
203/203 - 1s - loss: 0.6292 - accuracy: 0.8261 - val_loss: 0.6358 - val_accur
acy: 0.8385
Epoch 8/50
203/203 - 1s - loss: 0.6110 - accuracy: 0.8411 - val_loss: 0.5953 - val_accur
acy: 0.8547
Epoch 9/50
203/203 - 1s - loss: 0.5981 - accuracy: 0.8471 - val_loss: 0.5850 - val_accur
acy: 0.8588
Epoch 10/50
203/203 - 1s - loss: 0.5865 - accuracy: 0.8545 - val_loss: 0.5821 - val_accur
acy: 0.8594
Epoch 11/50
203/203 - 1s - loss: 0.5697 - accuracy: 0.8647 - val_loss: 0.5663 - val_accur
acy: 0.8657
Epoch 12/50
203/203 - 1s - loss: 0.5665 - accuracy: 0.8692 - val_loss: 0.5548 - val_accur
acy: 0.8704
Epoch 13/50
203/203 - 1s - loss: 0.5567 - accuracy: 0.8712 - val_loss: 0.5525 - val_accur
acy: 0.8727
Epoch 14/50
203/203 - 1s - loss: 0.5491 - accuracy: 0.8781 - val_loss: 0.5445 - val_accur
acy: 0.8733
Epoch 15/50
203/203 - 1s - loss: 0.5364 - accuracy: 0.8798 - val_loss: 0.5501 - val_accur
acy: 0.8744
Epoch 16/50
203/203 - 1s - loss: 0.5338 - accuracy: 0.8826 - val_loss: 0.5416 - val_accur
acy: 0.8802
Epoch 17/50
203/203 - 1s - loss: 0.5303 - accuracy: 0.8846 - val_loss: 0.5433 - val_accur
acy: 0.8773
Epoch 18/50
203/203 - 1s - loss: 0.5212 - accuracy: 0.8926 - val_loss: 0.5322 - val_accur
acy: 0.8779
Epoch 19/50
203/203 - 1s - loss: 0.5166 - accuracy: 0.8892 - val_loss: 0.5128 - val_accur
acy: 0.8900
```

```
Epoch 20/50
203/203 - 1s - loss: 0.5153 - accuracy: 0.8917 - val_loss: 0.5332 - val_accur
acy: 0.8744
Epoch 21/50
203/203 - 1s - loss: 0.5083 - accuracy: 0.8984 - val_loss: 0.5122 - val_accur
acy: 0.8918
Epoch 22/50
203/203 - 1s - loss: 0.5017 - accuracy: 0.9042 - val_loss: 0.5040 - val_accur
acy: 0.8924
Epoch 23/50
203/203 - 1s - loss: 0.4935 - accuracy: 0.9024 - val_loss: 0.5075 - val_accur
acy: 0.8866
Epoch 24/50
203/203 - 1s - loss: 0.4928 - accuracy: 0.9033 - val_loss: 0.5027 - val_accur
acy: 0.8958
Epoch 25/50
203/203 - 1s - loss: 0.4904 - accuracy: 0.9033 - val_loss: 0.5013 - val_accur
acy: 0.8941
Epoch 26/50
203/203 - 1s - loss: 0.4814 - accuracy: 0.9061 - val_loss: 0.5011 - val_accur
acy: 0.8964
Epoch 27/50
203/203 - 1s - loss: 0.4804 - accuracy: 0.9096 - val_loss: 0.4856 - val_accur
acy: 0.8993
Epoch 28/50
203/203 - 1s - loss: 0.4734 - accuracy: 0.9101 - val_loss: 0.4880 - val_accur
acy: 0.8987
Epoch 29/50
203/203 - 1s - loss: 0.4812 - accuracy: 0.9055 - val_loss: 0.4870 - val_accur
acy: 0.8999
Epoch 30/50
203/203 - 1s - loss: 0.4706 - accuracy: 0.9132 - val_loss: 0.4903 - val_accur
acy: 0.9005
Epoch 31/50
203/203 - 1s - loss: 0.4687 - accuracy: 0.9135 - val_loss: 0.4772 - val_accur
acy: 0.9010
Epoch 32/50
203/203 - 1s - loss: 0.4693 - accuracy: 0.9124 - val_loss: 0.4935 - val_accur
acy: 0.8941
Epoch 33/50
203/203 - 1s - loss: 0.4603 - accuracy: 0.9171 - val_loss: 0.4789 - val_accur
acy: 0.9051
Epoch 34/50
203/203 - 1s - loss: 0.4586 - accuracy: 0.9116 - val_loss: 0.4789 - val_accur
acy: 0.8981
Epoch 35/50
203/203 - 1s - loss: 0.4540 - accuracy: 0.9189 - val_loss: 0.4678 - val_accur
acy: 0.9051
Epoch 36/50
203/203 - 1s - loss: 0.4614 - accuracy: 0.9124 - val_loss: 0.4707 - val_accur
acy: 0.9045
Epoch 37/50
203/203 - 1s - loss: 0.4486 - accuracy: 0.9240 - val_loss: 0.4662 - val_accur
acy: 0.9010
Epoch 38/50
203/203 - 1s - loss: 0.4464 - accuracy: 0.9266 - val_loss: 0.4666 - val_accur
acy: 0.9045
```

```
Epoch 39/50
203/203 - 1s - loss: 0.4440 - accuracy: 0.9209 - val_loss: 0.4593 - val_accur
acy: 0.9045
Epoch 40/50
203/203 - 1s - loss: 0.4383 - accuracy: 0.9300 - val_loss: 0.4553 - val_accur
acy: 0.9068
Epoch 41/50
203/203 - 1s - loss: 0.4389 - accuracy: 0.9291 - val_loss: 0.4627 - val_accur
acy: 0.9045
Epoch 42/50
203/203 - 1s - loss: 0.4359 - accuracy: 0.9262 - val_loss: 0.4761 - val_accur
acy: 0.8918
Epoch 43/50
203/203 - 1s - loss: 0.4420 - accuracy: 0.9240 - val_loss: 0.4512 - val_accur
acy: 0.9086
Epoch 44/50
203/203 - 1s - loss: 0.4348 - accuracy: 0.9243 - val_loss: 0.4501 - val_accur
acy: 0.9080
Epoch 45/50
203/203 - 1s - loss: 0.4301 - accuracy: 0.9282 - val_loss: 0.4569 - val_accur
acy: 0.9109
Epoch 46/50
203/203 - 1s - loss: 0.4313 - accuracy: 0.9257 - val_loss: 0.4471 - val_accur
acy: 0.9126
Epoch 47/50
203/203 - 1s - loss: 0.4256 - accuracy: 0.9311 - val_loss: 0.4520 - val_accur
acy: 0.9080
Epoch 48/50
203/203 - 1s - loss: 0.4260 - accuracy: 0.9297 - val_loss: 0.4494 - val_accur
acy: 0.9115
Epoch 49/50
203/203 - 1s - loss: 0.4230 - accuracy: 0.9320 - val_loss: 0.4610 - val_accur
acy: 0.9062
Epoch 50/50
203/203 - 1s - loss: 0.4182 - accuracy: 0.9320 - val_loss: 0.4477 - val_accur
acy: 0.9120
6/6 [==============================] - 0s 3ms/step - loss: 0.9994 - accuracy:
0.5943
Test loss, Test accuracy: [0.9993738532066345, 0.5942857265472412]
```

PART 2: Convolutional Neural Network (CNN)

Task 1: *2D CNN.*

Have a look at the model below and answer the following:

- How many layers does it have?
- How many filter in each layer?
- Would the number of parmaters be similar to a fully connected NN?
- Is this specific NN performing regularization?

In [25]:
```
def get_net(input_shape,drop,dropRate,reg):
    #Defining the network architecture:
    model = Sequential()
    model.add(Permute((1,2,3),input_shape = input_shape))
    model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same', activation
='relu',name='Conv2D_1',kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activatio
n='relu',name='Conv2D_2',kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same', activatio
n='relu',name='Conv2D_3',kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activatio
n='relu',name='Conv2D_4',kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same', activatio
n='relu',name='Conv2D_5',kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    #Fully connected network tail:
    model.add(Dense(512, activation='elu',name='FCN_1'))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(Dense(128, activation='elu',name='FCN_2'))
    model.add(Dense(4, activation= 'softmax',name='FCN_3'))
    model.summary()
    return model
```

In [26]:
```
input_shape = (32,32,1)
learn_rate = 1e-5
decay = 1e-03
batch_size = 64
epochs = 25
drop = True
dropRate = 0.3
reg = 1e-2
NNet = get_net(input_shape,drop,dropRate,reg)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
permute (Permute)            (None, 32, 32, 1)         0         
_________________________________________________________________
Conv2D_1 (Conv2D)            (None, 32, 32, 64)        640       
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 64)        0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0         
_________________________________________________________________
Conv2D_2 (Conv2D)            (None, 16, 16, 128)       73856     
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       64        
_________________________________________________________________
Conv2D_3 (Conv2D)            (None, 16, 16, 128)       147584    
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
batch_normalization_5 (Batch (None, 16, 16, 128)       64        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 128)         0         
_________________________________________________________________
Conv2D_4 (Conv2D)            (None, 8, 8, 256)         295168    
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 256)         32        
_________________________________________________________________
Conv2D_5 (Conv2D)            (None, 8, 8, 256)         590080    
_________________________________________________________________
dropout_4 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 8, 8, 256)         32        
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 256)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0         
_________________________________________________________________
FCN_1 (Dense)                (None, 512)               2097664   
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0         
_________________________________________________________________
FCN_2 (Dense)                (None, 128)               65664     
_________________________________________________________________
FCN_3 (Dense)                (None, 4)                 516       
=================================================================
Total params: 3,271,492
Trainable params: 3,271,332
Non-trainable params: 160
_________________________________________________________________
```

In [27]:
```
NNet=get_net(input_shape,drop,dropRate,reg)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
permute_1 (Permute)          (None, 32, 32, 1)         0         
_________________________________________________________________
Conv2D_1 (Conv2D)            (None, 32, 32, 64)        640       
_________________________________________________________________
dropout_6 (Dropout)          (None, 32, 32, 64)        0         
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 32, 64)        128       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0         
_________________________________________________________________
Conv2D_2 (Conv2D)            (None, 16, 16, 128)       73856     
_________________________________________________________________
dropout_7 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 16, 16, 128)       64        
_________________________________________________________________
Conv2D_3 (Conv2D)            (None, 16, 16, 128)       147584    
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
batch_normalization_10 (Batc (None, 16, 16, 128)       64        
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 128)         0         
_________________________________________________________________
Conv2D_4 (Conv2D)            (None, 8, 8, 256)         295168    
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 256)         32        
_________________________________________________________________
Conv2D_5 (Conv2D)            (None, 8, 8, 256)         590080    
_________________________________________________________________
dropout_10 (Dropout)         (None, 8, 8, 256)         0         
_________________________________________________________________
batch_normalization_12 (Batc (None, 8, 8, 256)         32        
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 256)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 4096)              0         
_________________________________________________________________
FCN_1 (Dense)                (None, 512)               2097664   
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0         
_________________________________________________________________
FCN_2 (Dense)                (None, 128)               65664     
_________________________________________________________________
FCN_3 (Dense)                (None, 4)                 516       
=================================================================
Total params: 3,271,492
Trainable params: 3,271,332
Non-trainable params: 160
_________________________________________________________________
```

In [28]:
```
from tensorflow.keras.optimizers import *
import os
from tensorflow.keras.callbacks import *

#Defining the optimizar parameters:
AdamOpt = Adam(lr=learn_rate,decay=decay)

#Compile the network:
NNet.compile(optimizer=AdamOpt, metrics=['acc'], loss='categorical_crossentrop
y')

#Saving checkpoints during training:
Checkpath = os.getcwd()
Checkp = ModelCheckpoint(Checkpath, monitor='val_acc', verbose=1, save_best_on
ly=True, save_weights_only=True, save_freq=1)
```

In [29]:
```
#Preforming the training by using fit
# IMPORTANT NOTE: This will take a few minutes!
h = NNet.fit(x=BaseX_train, y=BaseY_train, batch_size=batch_size, epochs=epoch
s, verbose=1, validation_split=0, validation_data = (BaseX_val, BaseY_val), sh
uffle=True)
#NNet.save(model_fn)
```

```
Epoch 1/25
102/102 [==============================] - 18s 162ms/step - loss: 8.5007 - ac
c: 0.3504 - val_loss: 7.9215 - val_acc: 0.2500
Epoch 2/25
102/102 [==============================] - 17s 163ms/step - loss: 7.7052 - ac
c: 0.5118 - val_loss: 8.0498 - val_acc: 0.2512
Epoch 3/25
102/102 [==============================] - 17s 165ms/step - loss: 7.5296 - ac
c: 0.5614 - val_loss: 8.0904 - val_acc: 0.2569
Epoch 4/25
102/102 [==============================] - 17s 164ms/step - loss: 7.4107 - ac
c: 0.5819 - val_loss: 8.1286 - val_acc: 0.2714
Epoch 5/25
102/102 [==============================] - 16s 159ms/step - loss: 7.2934 - ac
c: 0.6304 - val_loss: 8.0654 - val_acc: 0.3056
Epoch 6/25
102/102 [==============================] - 16s 159ms/step - loss: 7.1918 - ac
c: 0.6543 - val_loss: 7.8972 - val_acc: 0.3345
Epoch 7/25
102/102 [==============================] - 16s 157ms/step - loss: 7.1311 - ac
c: 0.6744 - val_loss: 7.8012 - val_acc: 0.3640
Epoch 8/25
102/102 [==============================] - 16s 159ms/step - loss: 7.0946 - ac
c: 0.6798 - val_loss: 7.7292 - val_acc: 0.4005
Epoch 9/25
102/102 [==============================] - 16s 157ms/step - loss: 7.0324 - ac
c: 0.7029 - val_loss: 7.6802 - val_acc: 0.4062
Epoch 10/25
102/102 [==============================] - 16s 158ms/step - loss: 6.9823 - ac
c: 0.7114 - val_loss: 7.7079 - val_acc: 0.4051
Epoch 11/25
102/102 [==============================] - 16s 158ms/step - loss: 6.9345 - ac
c: 0.7223 - val_loss: 7.6792 - val_acc: 0.4051
Epoch 12/25
102/102 [==============================] - 16s 156ms/step - loss: 6.8930 - ac
c: 0.7309 - val_loss: 7.6453 - val_acc: 0.4120
Epoch 13/25
102/102 [==============================] - 16s 157ms/step - loss: 6.8857 - ac
c: 0.7260 - val_loss: 7.6553 - val_acc: 0.3999
Epoch 14/25
102/102 [==============================] - 16s 156ms/step - loss: 6.8302 - ac
c: 0.7549 - val_loss: 7.6467 - val_acc: 0.3987
Epoch 15/25
102/102 [==============================] - 16s 157ms/step - loss: 6.7913 - ac
c: 0.7571 - val_loss: 7.6402 - val_acc: 0.3762
Epoch 16/25
102/102 [==============================] - 16s 159ms/step - loss: 6.7759 - ac
c: 0.7583 - val_loss: 7.6077 - val_acc: 0.3889
Epoch 17/25
102/102 [==============================] - 16s 157ms/step - loss: 6.7342 - ac
c: 0.7663 - val_loss: 7.6194 - val_acc: 0.3808
Epoch 18/25
102/102 [==============================] - 16s 157ms/step - loss: 6.6998 - ac
c: 0.7711 - val_loss: 7.5811 - val_acc: 0.3860
Epoch 19/25
102/102 [==============================] - 16s 157ms/step - loss: 6.6817 - ac
c: 0.7682 - val_loss: 7.5979 - val_acc: 0.3773
```

```
Epoch 20/25
102/102 [==============================] - 16s 156ms/step - loss: 6.6465 - ac
c: 0.7861 - val_loss: 7.6038 - val_acc: 0.3791
Epoch 21/25
102/102 [==============================] - 16s 156ms/step - loss: 6.6231 - ac
c: 0.7832 - val_loss: 7.5938 - val_acc: 0.3785
Epoch 22/25
102/102 [==============================] - 16s 159ms/step - loss: 6.5786 - ac
c: 0.7937 - val_loss: 7.6143 - val_acc: 0.3727
Epoch 23/25
102/102 [==============================] - 16s 157ms/step - loss: 6.5530 - ac
c: 0.8055 - val_loss: 7.5845 - val_acc: 0.3762
Epoch 24/25
102/102 [==============================] - 16s 157ms/step - loss: 6.5499 - ac
c: 0.8004 - val_loss: 7.5682 - val_acc: 0.3773
Epoch 25/25
102/102 [==============================] - 16s 159ms/step - loss: 6.4970 - ac
c: 0.8132 - val_loss: 7.5504 - val_acc: 0.3808
```

In [30]:
```
# NNet.load_weights('Weights_1.h5')
```

In [31]:
```
results = NNet.evaluate(X_test,Y_test)
print('test loss, test acc:', results)
```

```
6/6 [==============================] - 0s 18ms/step - loss: 7.9150 - acc: 0.3
371
test loss, test acc: [7.9150285720825195, 0.33714285492897034]
```

Task 2: *Number of filters*

Rebuild the function `get_net` to have as an input argument a list of number of filters in each layers, i.e. for the CNN defined above the input should have been `[64, 128, 128, 256, 256]` . Now train the model with the number of filters reduced by half. What were the results.

In [32]:
```
#--------------------------Impelment your code here:---------------------------
-----------
def get_net(input_shape, drop, dropRate, reg, filters):
    #Defining the network architecture:
    model = Sequential()
    model.add(Permute((1, 2, 3), input_shape=input_shape))
    model.add(Conv2D(filters=filters[0], kernel_size=(3, 3), padding='same', a
ctivation='relu', name='Conv2D_1', kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=filters[1], kernel_size=(3, 3), padding='same', a
ctivation='relu', name='Conv2D_2', kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(Conv2D(filters=filters[2], kernel_size=(3, 3), padding='same', a
ctivation='relu',name='Conv2D_3',kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=filters[3], kernel_size=(3, 3), padding='same', a
ctivation='relu',name='Conv2D_4', kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(Conv2D(filters=filters[4], kernel_size=(3, 3), padding='same', a
ctivation='relu',name='Conv2D_5', kernel_regularizer=regularizers.l2(reg)))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(BatchNormalization(axis=1))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    #Fully connected network tail:
    model.add(Dense(512, activation='elu',name='FCN_1'))
    if drop:
        model.add(Dropout(rate=dropRate))
    model.add(Dense(128, activation='elu',name='FCN_2'))
    model.add(Dense(4, activation= 'softmax',name='FCN_3'))
    model.summary()
    return model

filters = [32, 64, 64, 128, 128]
NNet_new=get_net(input_shape,drop,dropRate,reg,filters)
NNet_new.compile(optimizer=AdamOpt, metrics=['acc'], loss='categorical_crossen
tropy')
h_new = NNet_new.fit(x=BaseX_train, y=BaseY_train, batch_size=batch_size, epoc
hs=epochs, verbose=1, validation_split=0, validation_data=(BaseX_val, BaseY_va
l), shuffle=True)
results_metrics = NNet_new.evaluate(X_test,Y_test)
print('Test loss, Test acc:', results_metrics)
#------------------------------------------------------------------------------
-----------
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
permute_2 (Permute)          (None, 32, 32, 1)         0         
_________________________________________________________________
Conv2D_1 (Conv2D)            (None, 32, 32, 32)        320       
_________________________________________________________________
dropout_12 (Dropout)         (None, 32, 32, 32)        0         
_________________________________________________________________
batch_normalization_13 (Batc (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
Conv2D_2 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
dropout_13 (Dropout)         (None, 16, 16, 64)        0         
_________________________________________________________________
batch_normalization_14 (Batc (None, 16, 16, 64)        64        
_________________________________________________________________
Conv2D_3 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
dropout_14 (Dropout)         (None, 16, 16, 64)        0         
_________________________________________________________________
batch_normalization_15 (Batc (None, 16, 16, 64)        64        
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
Conv2D_4 (Conv2D)            (None, 8, 8, 128)         73856     
_________________________________________________________________
dropout_15 (Dropout)         (None, 8, 8, 128)         0         
_________________________________________________________________
batch_normalization_16 (Batc (None, 8, 8, 128)         32        
_________________________________________________________________
Conv2D_5 (Conv2D)            (None, 8, 8, 128)         147584    
_________________________________________________________________
dropout_16 (Dropout)         (None, 8, 8, 128)         0         
_________________________________________________________________
batch_normalization_17 (Batc (None, 8, 8, 128)         32        
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 2048)              0         
_________________________________________________________________
FCN_1 (Dense)                (None, 512)               1049088   
_________________________________________________________________
dropout_17 (Dropout)         (None, 512)               0         
_________________________________________________________________
FCN_2 (Dense)                (None, 128)               65664     
_________________________________________________________________
FCN_3 (Dense)                (None, 4)                 516       
=================================================================
Total params: 1,392,772
Trainable params: 1,392,612
Non-trainable params: 160
_________________________________________________________________
Epoch 1/25
```

```
102/102 [==============================] - 9s 78ms/step - loss: 4.9187 - acc:
0.3881 - val_loss: 4.6354 - val_acc: 0.2500
Epoch 2/25
102/102 [==============================] - 8s 75ms/step - loss: 4.4907 - acc:
0.5206 - val_loss: 4.6999 - val_acc: 0.2564
Epoch 3/25
102/102 [==============================] - 8s 76ms/step - loss: 4.3440 - acc:
0.5581 - val_loss: 4.7265 - val_acc: 0.2512
Epoch 4/25
102/102 [==============================] - 8s 76ms/step - loss: 4.3172 - acc:
0.5594 - val_loss: 4.7363 - val_acc: 0.2564
Epoch 5/25
102/102 [==============================] - 8s 76ms/step - loss: 4.2809 - acc:
0.5804 - val_loss: 4.6924 - val_acc: 0.2506
Epoch 6/25
102/102 [==============================] - 8s 76ms/step - loss: 4.2296 - acc:
0.5960 - val_loss: 4.6012 - val_acc: 0.3050
Epoch 7/25
102/102 [==============================] - 8s 76ms/step - loss: 4.1942 - acc:
0.6107 - val_loss: 4.5395 - val_acc: 0.3461
Epoch 8/25
102/102 [==============================] - 8s 77ms/step - loss: 4.1713 - acc:
0.6168 - val_loss: 4.4959 - val_acc: 0.3605
Epoch 9/25
102/102 [==============================] - 8s 76ms/step - loss: 4.1553 - acc:
0.6185 - val_loss: 4.4879 - val_acc: 0.3634
Epoch 10/25
102/102 [==============================] - 8s 76ms/step - loss: 4.1295 - acc:
0.6235 - val_loss: 4.4830 - val_acc: 0.3628
Epoch 11/25
102/102 [==============================] - 8s 76ms/step - loss: 4.0983 - acc:
0.6329 - val_loss: 4.4814 - val_acc: 0.3605
Epoch 12/25
102/102 [==============================] - 8s 76ms/step - loss: 4.0771 - acc:
0.6359 - val_loss: 4.4815 - val_acc: 0.3623
Epoch 13/25
102/102 [==============================] - 8s 78ms/step - loss: 4.0548 - acc:
0.6397 - val_loss: 4.4844 - val_acc: 0.3617
Epoch 14/25
102/102 [==============================] - 8s 76ms/step - loss: 4.0624 - acc:
0.6422 - val_loss: 4.4786 - val_acc: 0.3634
Epoch 15/25
102/102 [==============================] - 8s 77ms/step - loss: 4.0375 - acc:
0.6495 - val_loss: 4.4947 - val_acc: 0.3588
Epoch 16/25
102/102 [==============================] - 8s 77ms/step - loss: 4.0205 - acc:
0.6460 - val_loss: 4.4989 - val_acc: 0.3524
Epoch 17/25
102/102 [==============================] - 8s 77ms/step - loss: 4.0045 - acc:
0.6662 - val_loss: 4.5160 - val_acc: 0.3466
Epoch 18/25
102/102 [==============================] - 8s 77ms/step - loss: 3.9997 - acc:
0.6587 - val_loss: 4.5171 - val_acc: 0.3461
Epoch 19/25
102/102 [==============================] - 8s 78ms/step - loss: 3.9774 - acc:
0.6745 - val_loss: 4.5293 - val_acc: 0.3414
Epoch 20/25
```

```
102/102 [==============================] - 8s 77ms/step - loss: 3.9584 - acc:
0.6731 - val_loss: 4.5325 - val_acc: 0.3426
Epoch 21/25
102/102 [==============================] - 8s 77ms/step - loss: 3.9694 - acc:
0.6664 - val_loss: 4.5348 - val_acc: 0.3420
Epoch 22/25
102/102 [==============================] - 8s 77ms/step - loss: 3.9771 - acc:
0.6802 - val_loss: 4.5379 - val_acc: 0.3426
Epoch 23/25
102/102 [==============================] - 8s 77ms/step - loss: 3.9568 - acc:
0.6741 - val_loss: 4.5434 - val_acc: 0.3409
Epoch 24/25
102/102 [==============================] - 8s 78ms/step - loss: 3.9172 - acc:
0.6923 - val_loss: 4.5483 - val_acc: 0.3391
Epoch 25/25
102/102 [==============================] - 8s 77ms/step - loss: 3.9014 - acc:
0.6944 - val_loss: 4.5590 - val_acc: 0.3374
6/6 [==============================] - 0s 11ms/step - loss: 4.9387 - acc: 0.2
343
Test loss, Test acc: [4.9386749267578125, 0.23428571224212646]
```

That's all folks! See you :)