Software implementation plan

* I am using **Tensorflow** and **python 3** in google colab for the training of the model.As we are still in the reducing the parameters stage to improve the model after reducing the model we need to convert the model to verilog using **H4ml** and use the **FPGA board** to deploy

### First i have implemented the Mobilenet v1 from the original paper and their architecture

def mobileNet(input\_shape, n\_classes):

"""

Manish's variation of MobileNetV1

"""

def mobilenet\_block(x, f, s=1):

x = tf.keras.layers.DepthwiseConv2D(3, strides=s, padding='same')(x)

x = tf.keras.layers.BatchNormalization()(x)

x = tf.keras.layers.Activation('relu')(x)

x = tf.keras.layers.Conv2D(f, 1, strides=1, padding='same')(x)

x = tf.keras.layers.BatchNormalization()(x)

x =tf.keras.layers.Activation('relu')(x)

return x

input = tf.keras.layers.Input(input\_shape)

x = tf.keras.layers.Conv2D(32, 3, strides=2, padding='same')(input)

x = tf.keras.layers.BatchNormalization()(x)

x = tf.keras.layers.Activation('relu')(x)

x = mobilenet\_block(x, 64)

x = mobilenet\_block(x, 128)

x = mobilenet\_block(x, 128)

x = mobilenet\_block(x, 256)

x = mobilenet\_block(x, 256)

x = mobilenet\_block(x, 512)

for \_ in range(5):

x = mobilenet\_block(x, 512)

x = mobilenet\_block(x, 1024)

x = mobilenet\_block(x, 1024)

x = tf.keras.layers.GlobalAvgPool2D()(x)

# x = tf.keras.layers.AveragePooling2D()(x)

output = tf.keras.layers.Dense(n\_classes, activation='softmax')(x)

model = tf.keras.models.Model(input, output)

return model

### High Level Autocorrelation:

In order to reduce the no of parameters we need to reduce the parameters of convolution layers in the mobilenet and we used HLAC to reduce the no of parameters in the conv2d layer

# fan\_in = height \* width

img\_size = 32 # 32 x 32

masks\_origin = [

"000010000", # 0th order HLAC, 1 := 3 / sqrt(h\*w), 0 := 3 / sqrt(h\*w)

"000011000", # 1st order HLAC

"001010000",

"010010000",

"100010000",

"000111000", # 2nd order HLAC

"001010100",

"010010010",

"100010001",

"001110000",

"010010100",

"100010010",

"000110001",

"000011100",

"001010010",

"010010001",

"100011000",

"010110000",

"100010100",

"000110010",

"000010101",

"000011010",

"001010001",

"010011000",

"101010000",

]

masks = []

masks\_n = []

c = 0

for mask\_bin in masks\_origin:

# if c % 3 == 0:

# m = []

s = 0

mask = []

for ch in mask\_bin:

if c % 3 == 0:

m = []

if int(ch) == 1:

s += 1

m.append([3 / np.sqrt(img\_size \* img\_size )])

else:

m.append([-3 /np.sqrt(img\_size \* img\_size )])

And we use the is used in the first layer of the conv2d

And also used kernel sharing and group kernel sharing and efficient weight sharing in kernels but all these implementations can be used to replace normal convolution but we need to reduce

### **Group sharing**

class Conv2DTiledKernel\_with\_groups(tf.keras.layers.Layer):

def \_\_init\_\_(self, filters, kernel\_size, multiplies, groups,\*\*kwargs):

self.filters = filters

self.kernel\_size = kernel\_size

self.multiplies = multiplies

self.groups=groups

super(Conv2DTiledKernel\_with\_groups, self).\_\_init\_\_(\*\*kwargs)

def build(self, input\_shape):

shape = list(self.kernel\_size) + [input\_shape[-1]//self.groups, self.filters//self.groups]

self.kernel = self.add\_weight(name='kernel', shape=shape,

initializer='glorot\_uniform')

self.kernel.\_trainable\_weights=False

super(Conv2DTiledKernel\_with\_groups, self).build(input\_shape)

def call(self, x):

mult = list(self.multiplies) + [1, self.groups]

kernel\_tiled = tf.tile(self.kernel, mult)

return tf.keras.backend.conv2d(x, kernel\_tiled)

def compute\_output\_shape(self, input\_shape):

return input\_shape[:-1] + (self.filters,)

### **Improving Pointwise convolution**

the pointwise convolution parameters as it is the major one causing the issue at deeper layers with larger number of filters so we are using technique of decomposing the multiplication of the

Pointwise convolution which can be done by using the **Improved pointwise convolution**

**def IPC\_block(input, f,alpha,k):**

**#depthwise convolution**

**x1 = tf.keras.layers.DepthwiseConv2D(3, strides=1, padding='same')(input)**

**y=input.shape**

**depth\_shape=x1.shape**

**channels=depth\_shape[-1]**

**x = tf.keras.layers.BatchNormalization()(x1)**

**x = tf.keras.layers.Activation('relu')(x)**

**#seed=np.random.uniform(size=y[-1],low=-1,high=1)**

**#seed=np.reshape(seed,(y[-1],1,1))**

**#seed=tf.convert\_to\_tensor(seed)**

**#alpha=tf.cast(alpha,'float64')**

**seed=tf.random.uniform(**

**(1,input.shape[-1],1,1), minval=-1, maxval=0, dtype=tf.dtypes.float32, seed=None, name=None)**

**# conv2d with k filters with (1-alpha)\*f kernel size and with input z**

**beta=tf.keras.layers.Conv2D(k,kernel\_size=[int((1-alpha)\*f)+1,1], strides=1, padding='same')(seed)**

**w11,x11,y1,z1=beta.shape**

**print(beta.shape)**

**beta=tf.reshape(beta,(1,1,(w11\*x11\*y1\*z1),1))**

**#point wise conv2d with (1-alpha)\*f filters and input x1**

**x2=tf.keras.layers.Conv2D(int((1-alpha)\*f),1,strides=1,padding='same')(input)**

**print(" point wise convolution with dimensions as channels in input and int(1-alpha)\*f",x2.shape)**

**w22,x12,y2,z2=x2.shape**

**x2=tf.reshape(x2,(1,1,(x12\*y2\*z2),1))**

**y1 = tf.keras.layers.Concatenate(axis=2)([beta, x2])**

**\_,t1,t2,t3=y1.shape**

**y1=tf.reshape(y1,(1,1,(t1\*t2\*t3)//32,32))**

**print("after concatenating",y1.shape)**

**x = tf.keras.layers.BatchNormalization()(y1)**

**x =tf.keras.layers.Activation('relu')(x)**

**return x**

We use this function instead of the point wise convolution which can be used to reduce the parameters.In order to maintain the consistency in the accuracy we are changing only last 4 block of the point wise convolution

### **Result:**

There is a problem with the Improved Pointwise convolution as we are doing the reshape and concate the conv2d is seeing only the channels of the inputs but there are more values in the 2nd channel so need to check the dimensions of the concatenate.

I will be working on the **improved Pointwise convolution** and all the changes can be viewed at [google colab link](https://colab.research.google.com/drive/1aek1xR4lT55baW2wTJhP28Oz_sPrwBdQ#scrollTo=CcNV9NwxeiwE)