## Amir Hossein Eyvazkhani - 1747696

Ex 05

## Task 2

The coverage is checked in Collab. It reaches more than 92 percent accuracy and less than 0.22 loss in the first 20 epochs. In my local machine, I did not have any compute power for showing the output. Sorry about that...

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import numpy as np
import matplotlib.pyplot as plt

class MyBatchNormalization(keras.layers.Layer):
    def __init__(
        self,
        epsilon=1e-5
    ):
        super().__init__()
        self.epsilon = epsilon


    def build(self, input_shape):
        param_shape = (input_shape[-1],)
        self.gamma = self.add_weight(
                name="gamma",
                shape=param_shape,
                dtype='float32',
                trainable=True,
            )
        self.beta = self.add_weight(
                name="beta",
                shape=param_shape,
                dtype='float32',
                trainable=True,
        )
        self.built = True
```

```python
    def call(self, inputs):
        batch_mean = tf.reduce_mean(inputs, axis=0)
        batch_variance = tf.math.reduce_std(inputs, axis=0)
        normalized_inputs = (inputs - batch_mean) / tf.sqrt(batch_variance + self.epsilon)
        outputs = self.gamma * normalized_inputs + self.beta
        return outputs


# for the last part of the questuion
class MyBatchNormalization_joint_version(keras.layers.Layer):
    def __init__(
        self,
        epsilon=1e-5
    ):
        super().__init__()
        self.epsilon = epsilon


    def build(self, input_shape):
        self.gamma = self.add_weight(
                name="gamma",
                shape=(1,),
                dtype='float32',
                trainable=True,
            )
        self.beta = self.add_weight(
                name="beta",
                shape=(1,),
                dtype='float32',
                trainable=True,
        )
        self.built = True

    def call(self, inputs):
        batch_mean = tf.reduce_mean(inputs)
        batch_variance = tf.math.reduce_std(inputs)
        normalized_inputs = (inputs - batch_mean) / tf.sqrt(batch_variance + self.epsilon)
        outputs = self.gamma * normalized_inputs + self.beta
        return outputs


def define_model():
    inputs = keras.Input(shape=(28,28,1))

    K = 20 # number of convolution layers per block
    L = 3  # number of blocks
    x = inputs
    for i in range(0,L):
        for j in range(0,K):
            x = layers.Conv2D(32, 3, activation="relu",padding="same")(x)
            x = MyBatchNormalization()(x)
            # for the second part we should use this version of BatchNormalization
            # x = MyBatchNormalization_joint_version()(x)
        x = layers.MaxPooling2D(3)(x)
    x = layers.GlobalMaxPooling2D()(x)
    outputs = layers.Dense(10,activation='softmax')(x)
    model = keras.Model(inputs,outputs)
    # model.summary()
    return model
```

In [ ]:

```
# Load and preprocess training data (Fashion-MNIST)
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)

# Define and train model
model = define_model()
model.compile(loss=keras.losses.CategoricalCrossentropy(),optimizer=keras.optimizers.Adam(),metrics=["accuracy"])
model.fit(train_images,train_labels, batch_size=64, epochs=100)
```

# Task 3

In [ ]:

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import numpy as np
import matplotlib.pyplot as plt

def define_model():
    # I define the implementation for the model manually here
    # in other words, it is not generalized with K, M, and L
    # and that is because I could not figure it out when we needed spatial sampling
    # also it was mentioned in the question to consider the graph for the implementation
    # the name of the layer has the following format: [LAYER_TYPE]_[L]_[K]
    # and each M = 2 layer a residual connection is added
    inputs = keras.Input(shape=(28,28,1))
    conv2d_0_0 = layers.Conv2D(32, 3, activation="relu", padding="same")(inputs)
    conv2d_0_1 = layers.Conv2D(32, 3, activation="relu", padding="same")(conv2d_0_0)
    matchChannel0 = layers.Conv2D(32, 1, padding="same")(inputs)
    add0 = layers.add([conv2d_0_1, matchChannel0])
    conv2d_0_2 = layers.Conv2D(32, 3, activation="relu", padding="same")(add0)
    conv2d_0_3 = layers.Conv2D(32, 3, activation="relu", padding="same")(conv2d_0_2)
    add1 = layers.add([conv2d_0_3, add0])
    maxPooling2D_1 = layers.MaxPooling2D(pool_size=(3,3), padding='valid', strides=(1,1))(add1)
    conv2d_1_0 = layers.Conv2D(32, 3, activation="relu", padding="same")(maxPooling2D_1)
    conv2d_1_1 = layers.Conv2D(32, 3, activation="relu", padding="same")(conv2d_1_0)
    averagePooling2D_1 = layers.MaxPooling2D(pool_size=(3,3), padding='valid', strides=(1,1))(add1)
    add2 =  layers.add([conv2d_1_1, averagePooling2D_1])
    conv2d_1_2 = layers.Conv2D(32, 3, activation="relu", padding="same")(add2)
    conv2d_1_3 = layers.Conv2D(32, 3, activation="relu", padding="same")(conv2d_1_2)
    add3 =  layers.add([conv2d_1_3, add2])
    maxPooling2D_2 = layers.MaxPooling2D(pool_size=(3,3), padding='valid', strides=(1,1))(add3)
    conv2d_2_0 = layers.Conv2D(32, 3, activation="relu", padding="same")(maxPooling2D_2)
    conv2d_2_1 = layers.Conv2D(32, 3, activation="relu", padding="same")(conv2d_2_0)
    averagePooling2D_2 = layers.MaxPooling2D(pool_size=(3,3), padding='valid', strides=(1,1))(add3)
    add4 =  layers.add([conv2d_2_1, averagePooling2D_2])
    conv2d_2_2 = layers.Conv2D(32, 3, activation="relu", padding="same")(add4)
    conv2d_2_3 = layers.Conv2D(32, 3, activation="relu", padding="same")(conv2d_2_2)
    add5 =  layers.add([conv2d_2_3, add4])
    falttening = layers.GlobalMaxPooling2D()(add5)
    outputs = layers.Dense(10,activation='softmax')(falttening)
    model = keras.Model(inputs,outputs)
    # model.summary()
    return model

# Load and preprocess training data (Fashion-MNIST)
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)
```

```python
# Define and train model
model = define_model()
model.compile(loss=keras.losses.CategoricalCrossentropy(),optimizer=keras.optimizers.Adam(),metrics=["accuracy"])
model.fit(train_images,train_labels, batch_size=64, epochs=5) # just tested with 5 epochs because of the limited comp
```

```
Epoch 1/5
WARNING:tensorflow:From h:\Uni\WiSe 2024\ML LAB\ml_lab_venv\Lib\site-packages\keras\src\utils\tf_utils.py:492: The na
me tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From h:\Uni\WiSe 2024\ML LAB\ml_lab_venv\Lib\site-packages\keras\src\engine\base_layer_utils.py:38
4: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_f
unctions instead.

938/938 [==============================] - 182s 187ms/step - loss: 0.6447 - accuracy: 0.7724
Epoch 2/5
938/938 [==============================] - 172s 183ms/step - loss: 0.3844 - accuracy: 0.8622
Epoch 3/5
938/938 [==============================] - 180s 192ms/step - loss: 0.3252 - accuracy: 0.8834
Epoch 4/5
938/938 [==============================] - 175s 187ms/step - loss: 0.2957 - accuracy: 0.8949
Epoch 5/5
938/938 [==============================] - 173s 185ms/step - loss: 0.2743 - accuracy: 0.9021
```

```
Out[ ]: <keras.src.callbacks.History at 0x204c6bb65d0>
```

It reaches suitable accuracy and loss without batchnorm.

```
In [ ]:
```