In [7]:

```
import os
import time
import shelve
import random
import numpy as np
import pandas as pd
import tensorflow as tf
from pandas import DataFrame
import matplotlib.pyplot as plt
```

In [8]:

```
def load_data(name):

    if name == 'mnist':
        (X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
    elif name == 'fashion_mnist':
        (X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.loa
d_data()
    else:
        print('Only mnist or fashion_mnist.')
        return False

    imageSize  = X_train.shape[1]*X_train.shape[2]
    numClasses = np.max(y_train)+1

    X_train = np.reshape(X_train.astype(float)/255.0, (-1, 784))
    X_test  = np.reshape(X_test.astype(float)/255.0, (-1, 784))

    y_train = tf.keras.utils.to_categorical(y_train, num_classes=numClasses)
    y_test  = tf.keras.utils.to_categorical(y_test, num_classes=numClasses)

    X_val   = X_train[-10000:]
    y_val   = y_train[-10000:]
    X_train = X_train[:-10000]
    y_train = y_train[:-10000]

    print('Data Split: ')
    print(f'X_train: {X_train.shape}, y_train: {y_train.shape}')
    print(f'X_test : {X_test.shape }, y_test : {y_test.shape }')
    print(f'X_val  : {X_val.shape  }, y_val  : {y_val.shape  }')

    data = {}
    data['X_train'] = X_train
    data['y_train'] = y_train
    data['X_val']   = X_val
    data['y_val']   = y_val
    data['X_test']  = X_test
    data['y_test']  = y_test

    data['imageSize'] = imageSize

    return data
```

In [9]:

```
class MLP(object):

    def __init__(self, name, size_input, size_hidden, size_output, learning_rate=
0.01, optimizer='SGD', weight_coeff=1,\
                 Reg=None, RegC=0, training=None, validation=None, accuracy=0, dev
ice=None):

        self.name          = name
        self.size_input    = size_input
        self.size_hidden   = size_hidden
        self.size_output   = size_output
        self.learning_rate = learning_rate
        self.optimizer     = optimizer
        self.Reg           = Reg
        self.RegC          = RegC
        self.training      = training
        self.validation    = validation
        self.accuracy      = accuracy
        self.device        = device
        self.weight_coeff  = weight_coeff

        self.W1 = self.initWeights(self.size_input, self.size_hidden[0], self.weig
ht_coeff)
        self.b1 = self.initWeights(1, self.size_hidden[0], self.weight_coeff)

        self.W2 = self.initWeights(self.size_hidden[0], self.size_hidden[1], self.
weight_coeff)
        self.b2 = self.initWeights(1, self.size_hidden[1], self.weight_coeff)

        self.W3 = self.initWeights(self.size_hidden[1], self.size_hidden[2], self.
weight_coeff)
        self.b3 = self.initWeights(1, self.size_hidden[2], self.weight_coeff)

        self.W4 = self.initWeights(self.size_hidden[2], self.size_output, self.wei
ght_coeff)
        self.b4 = self.initWeights(1, self.size_output, self.weight_coeff)

        self.varibles = [self.W1, self.b1, self.W2, self.b2, self.W3, self.b3, sel
f.W4, self.b4]


    def initWeights(self, rows, columns, multFactor=1):
        return tf.Variable(multFactor*tf.random.normal([rows, columns]))

    def forward(self, X):

        if self.device is not None:
            with tf.device('gpu:0' if self.device=='gpu' else 'cpu'):
                self.y = self.compute_output(X)
        else:
            self.y = self.compute_output(X)

        return self.y
    def getRegLoss(self, X_train):

        if self.Reg=='L2':
            return (self.RegC/X_train.shape[0])*(tf.reduce_sum(tf.math.square(self
.W1)) +
                                                  tf.reduce_sum(tf.math.square(self
.W2)) +
                                                  tf.reduce_sum(tf.math.square(self
.W3)) +
                                                  tf.reduce_sum(tf.math.square(self
.W4)))

        elif self.Reg=='L1':
            return (self.RegC/X_train.shape[0])*tf.abs(tf.reduce_sum(self.W1) +
                                                        tf.reduce_sum(self.W2) +
                                                        tf.reduce_sum(self.W3) +
                                                        tf.reduce_sum(self.W4))

        elif self.Reg=='L1+L2':
            L2 =  (self.RegC/X_train.shape[0])*(tf.reduce_sum(tf.math.square(self.
W1)) +
                                                 tf.reduce_sum(tf.math.square(self.
W2)) +
                                                 tf.reduce_sum(tf.math.square(self.
W3)) +
                                                 tf.reduce_sum(tf.math.square(self.
W4)))

            L1 = (self.RegC/X_train.shape[0])*tf.abs(tf.reduce_sum(self.W1) +
                                                      tf.reduce_sum(self.W2) +
                                                      tf.reduce_sum(self.W3) +
                                                      tf.reduce_sum(self.W4))
            return L1+L2

        else:
            return 0

    def loss(self, y_pred, y_true):

        y_true_tf = tf.cast(tf.reshape(y_true, (-1, self.size_output)), dtype=tf.f
loat32)
        y_pred_tf = tf.cast(y_pred, dtype=tf.float32)

        loss = tf.keras.losses.CategoricalCrossentropy()(y_true_tf, y_pred_tf)
        return loss

    def backward(self, X_train, y_train):

        if self.optimizer=='SGD':
            optimizer = tf.keras.optimizers.SGD(learning_rate=self.learning_rate)

        elif self.optimizer=='Adam':
            optimizer = tf.keras.optimizers.Adam(learning_rate=self.learning_rate)

        elif self.optimizer=='RMSProp':
            optimizer = tf.keras.optimizers.RMSprop(learning_rate=self.learning_ra
te)

        else:
            pass

        if self.Reg is not None and self.RegC==0:
            print('Regularization coffecient argument was 0, seeting it to default
lamda=0.01')
            self.RegC = 0.01;

        with tf.GradientTape() as tape:
            predicted = self.forward(X_train)
            current_loss = self.loss(predicted, y_train)
            current_loss += self.getRegLoss(X_train)

        grads = tape.gradient(current_loss, self.varibles)
        optimizer.apply_gradients(zip(grads, self.varibles))

    def compute_output(self, X):

        X_tf = tf.cast(X, dtype=tf.float32)

        w1Hat = tf.matmul(X_tf,  self.W1) + self.b1
        h1Hat = tf.nn.relu(w1Hat)

        w2Hat = tf.matmul(h1Hat, self.W2) + self.b2
        h2Hat = tf.nn.relu(w2Hat)

        w3Hat = tf.matmul(h2Hat, self.W3) + self.b3
        h3Hat = tf.nn.relu(w3Hat)

        w4Hat = tf.matmul(h3Hat, self.W4) + self.b4
        output = tf.nn.softmax(w4Hat)

        return output

    def getAccuracy(self, predictions, outputs):
        preds  = np.argmax(predictions, axis=1)
        y_true = np.argmax(outputs, axis=1)

        return (preds==y_true).mean()
```

In [10]:

```
def trainModel(model, data, NUM_EPOCHS=10, batchSize=50, seedVal=1234):

    X_train = data['X_train']
    y_train = data['y_train']
    X_val   = data['X_val']
    y_val   = data['y_val']

    training = np.zeros(shape=(NUM_EPOCHS, 3))
    validation = np.zeros(shape=(NUM_EPOCHS, 3))

    train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(batchS
ize)
    val_ds   = tf.data.Dataset.from_tensor_slices((X_val, y_val)).batch(batchSize)

    print(f'\n\n**************** Training model: {model.name} with optimizer: {mod
el.optimizer} and seed: {seedVal} ****************\n')
    time_start = time.time()
    for epoch in range(NUM_EPOCHS):
        train_loss = tf.zeros([1, 1], dtype=tf.float32)
        val_loss   = tf.zeros([1, 1], dtype=tf.float32)

        train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(
25, seed = epoch*(seedVal)).batch(batchSize)
        val_ds   = tf.data.Dataset.from_tensor_slices((X_val, y_val)).shuffle(25,
seed = epoch*(seedVal)).batch(batchSize)

        for inputs, outputs in train_ds:
            train_pred = model.forward(inputs)
            train_loss = train_loss + model.loss(train_pred, outputs)
            model.backward(inputs, outputs)
            train_acc = model.getAccuracy(train_pred, outputs)

        for inputs, outputs in val_ds:
            val_pred = model.forward(inputs)
            val_loss = val_loss + model.loss(val_pred, outputs)
            val_acc  = model.getAccuracy(val_pred, outputs)

        # train_loss = np.array(train_loss)
        # val_loss = np.array(val_loss)

        training[epoch] = [epoch+1, train_acc, np.sum(train_loss)/X_train.shape[0
]]
        validation[epoch] = [epoch+1, val_acc, np.sum(train_loss)/X_train.shape[0
]]

        print('# Epoch:={}/{}  - train loss:={:.4f} - val loss:={:.4f}, train acc:
={:.2f} - val acc:={:.2f}'\
              .format(epoch+1, NUM_EPOCHS, np.sum(train_loss)/X_train.shape[0], np
.sum(val_loss)/X_val.shape[0], train_acc, val_acc))

    time_taken = time.time()-time_start
    print(f'\nTotal time taken (in seconds): {time_taken: .2f}')
    print(f'\nFinished training model: {model.name}\n')
    model.training = training
    model.validation = validation

def testModel(model, data):

    X_test = data['X_test']
    y_test = data['y_test']

    preds = model.forward(X_test)

    pred = np.argmax(preds, axis=1)
    y_true= np.argmax(y_test, axis=1)

    model.accuracy = (pred==y_true).mean()*100

    print(f'**************** Testing ****************')
    print(f'{model.name} model accuracy = {model.accuracy:.2f}%')
    print(f'*****************************************')

def plotAccuracyAndLoss(model):

    training = model.training
    validation = model.validation
    fig, (ax1, ax2) = plt.subplots(1, 2)
    training[:, -1] = training[:, -1]/np.linalg.norm(training[:, -1])
    ax1.plot(training[:,0], training[:,1], 'g')
    ax1.plot(training[:,0], training[:,2], 'b')
    ax1.set_title('Training')
    ax1.legend(["Accuracy", "Loss"])

    validation[:, -1] = validation[:, -1]/np.linalg.norm(validation[:, -1])
    ax2.plot(validation[:,0], validation[:,1], 'g')
    ax2.plot(validation[:,0], validation[:,2], 'b')
    ax2.set_title('Validation')
    ax2.legend(["Accuracy", "Loss"])
    plt.show()
```

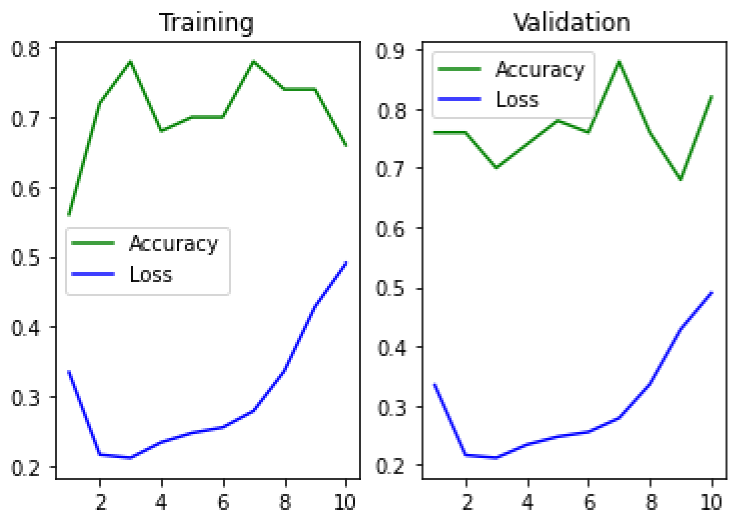
In [11]:

```
def main():

    for j in range(1):
        if j==1:
            data = load_data('mnist')
            size_hidden = [128, 128, 128]
            learning_rate = 5e-3
            weight_coeff  = 1e-2

        if j==0:
            data = load_data('fashion_mnist')
            size_hidden = [1024, 512, 256]
            learning_rate = 3e-4
            weight_coeff  = 1e-3

        for k in range(1):
            if k==0:
                opt = 'Adam'
            elif k==1:
                opt = 'RMSprop'
            elif k==2:
                opt = 'SGD'
            else:
                pass

            imageSize = data['imageSize']

            size_input  = imageSize
            size_output = 10

            allModels = {}
            allModels['mlp_on_gpu_default'] = {}
            allModels['mlp_on_gpu_RegL1']   = {}
            allModels['mlp_on_gpu_RegL2']   = {}

            for model_name in allModels:
                model = allModels[model_name]

                cnt = -1

                numEpochs = 10
                batchSize = 50
                numTrials = 10

                seeds = random.sample(range(1000, 9999), numTrials)

                # loss     = np.zeros(shape=(numEpochs, 1))
                accuracy = np.zeros(shape=(numTrials, 1))

                for i in seeds:
                    cnt += 1

                    np.random.seed(i)
                    tf.random.set_seed(i)

                    print(f'Count: {cnt}, j=: {j}')
                    if model_name == 'mlp_on_gpu_default':
                        model['name'] = MLP('mlp_on_gpu_default', size_input, size
_hidden, size_output, learning_rate, opt, weight_coeff,\
                                             device='gpu')

                    elif model_name == 'mlp_on_gpu_RegL1':
                        model['name'] = MLP('mlp_on_gpu_RegL1', size_input, size_h
idden, size_output, learning_rate, opt, weight_coeff,\
                                            'L1', 0.01, device='gpu')

                    elif model_name == 'mlp_on_gpu_RegL2':
                        model['name'] = MLP('mlp_on_gpu_RegL2', size_input, size_h
idden, size_output, learning_rate, opt, weight_coeff,\
                                            'L2', 0.01, device='gpu')

                    else:
                        pass

                    trainModel(model['name'], data, numEpochs, batchSize, i)
                    testModel(model['name'], data)

                    accuracy[cnt] = model['name'].accuracy

                    plotAccuracyAndLoss(model['name'])

                    allModels[model_name][i] = model['name']
                    allModels[model_name]['Accuracy'] = [np.mean(accuracy), np.var
(accuracy)]

        if j==0:
            mnist = allModels
        elif j==1:
            fashion_mnist = allModels
        else:
            pass



    return mnist, fashion_mnist
```

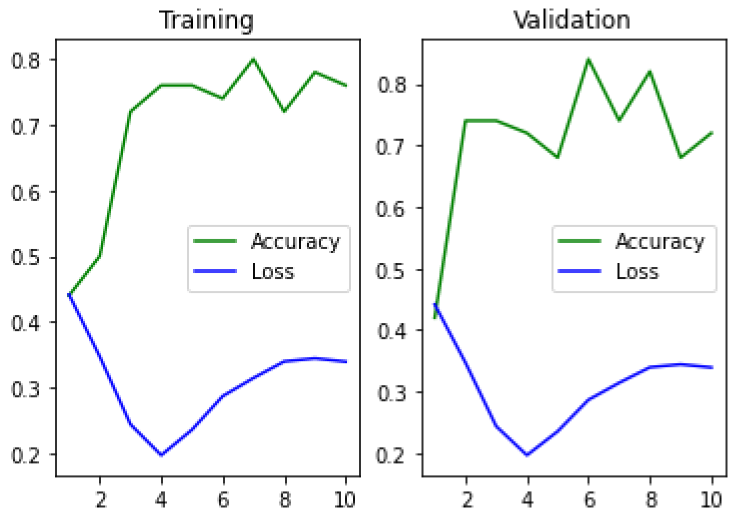
In [ ]:

```
if __name__ == "__main__":
    mnist, fashion_mnist = main()
```

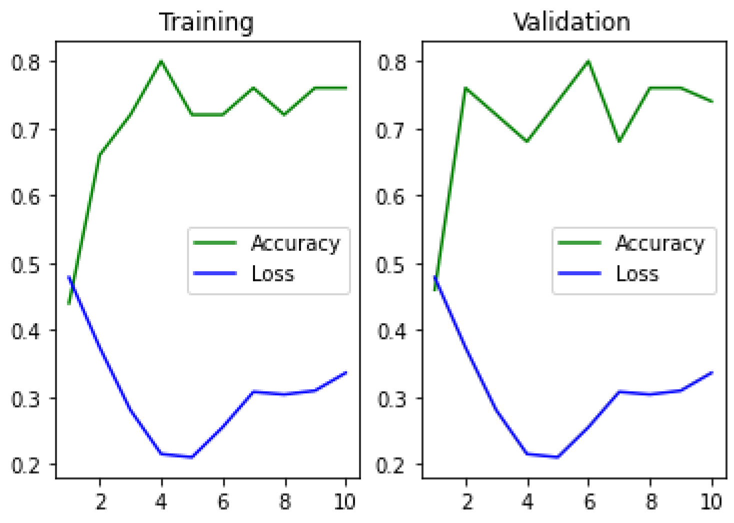
```
Data Split:
X_train: (50000, 784), y_train: (50000, 10)
X_test : (10000, 784), y_test : (10000, 10)
X_val  : (10000, 784), y_val  : (10000, 10)
Count: 0, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 1032 ****************

# Epoch:=1/10  - train loss:=0.0221 - val loss:=0.0152, train acc:=0.5
6 - val acc:=0.76
# Epoch:=2/10  - train loss:=0.0143 - val loss:=0.0136, train acc:=0.7
2 - val acc:=0.76
# Epoch:=3/10  - train loss:=0.0140 - val loss:=0.0143, train acc:=0.7
8 - val acc:=0.70
# Epoch:=4/10  - train loss:=0.0154 - val loss:=0.0180, train acc:=0.6
8 - val acc:=0.74
# Epoch:=5/10  - train loss:=0.0163 - val loss:=0.0155, train acc:=0.7
0 - val acc:=0.78
# Epoch:=6/10  - train loss:=0.0169 - val loss:=0.0167, train acc:=0.7
0 - val acc:=0.76
# Epoch:=7/10  - train loss:=0.0184 - val loss:=0.0193, train acc:=0.7
8 - val acc:=0.88
# Epoch:=8/10  - train loss:=0.0222 - val loss:=0.0298, train acc:=0.7
4 - val acc:=0.76
# Epoch:=9/10  - train loss:=0.0283 - val loss:=0.0467, train acc:=0.7
4 - val acc:=0.68
# Epoch:=10/10  - train loss:=0.0324 - val loss:=0.0276, train acc:=0.
66 - val acc:=0.82

Total time taken (in seconds):  181.15

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 84.52%
*****************************************
```



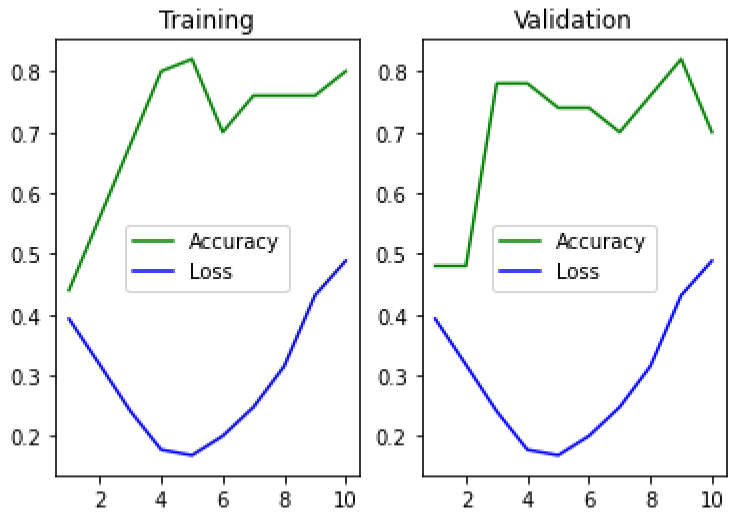
```
Count: 1, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 9657 ****************

# Epoch:=1/10  - train loss:=0.0267 - val loss:=0.0233, train acc:=0.4
4 - val acc:=0.42
# Epoch:=2/10  - train loss:=0.0210 - val loss:=0.0174, train acc:=0.5
0 - val acc:=0.74
# Epoch:=3/10  - train loss:=0.0147 - val loss:=0.0124, train acc:=0.7
2 - val acc:=0.74
# Epoch:=4/10  - train loss:=0.0119 - val loss:=0.0122, train acc:=0.7
6 - val acc:=0.72
# Epoch:=5/10  - train loss:=0.0142 - val loss:=0.0164, train acc:=0.7
6 - val acc:=0.68
# Epoch:=6/10  - train loss:=0.0173 - val loss:=0.0163, train acc:=0.7
4 - val acc:=0.84
# Epoch:=7/10  - train loss:=0.0190 - val loss:=0.0191, train acc:=0.8
0 - val acc:=0.74
# Epoch:=8/10  - train loss:=0.0205 - val loss:=0.0201, train acc:=0.7
2 - val acc:=0.82
# Epoch:=9/10  - train loss:=0.0208 - val loss:=0.0202, train acc:=0.7
8 - val acc:=0.68
# Epoch:=10/10  - train loss:=0.0205 - val loss:=0.0191, train acc:=0.
76 - val acc:=0.72

Total time taken (in seconds):  180.52

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 81.57%
*****************************************
```



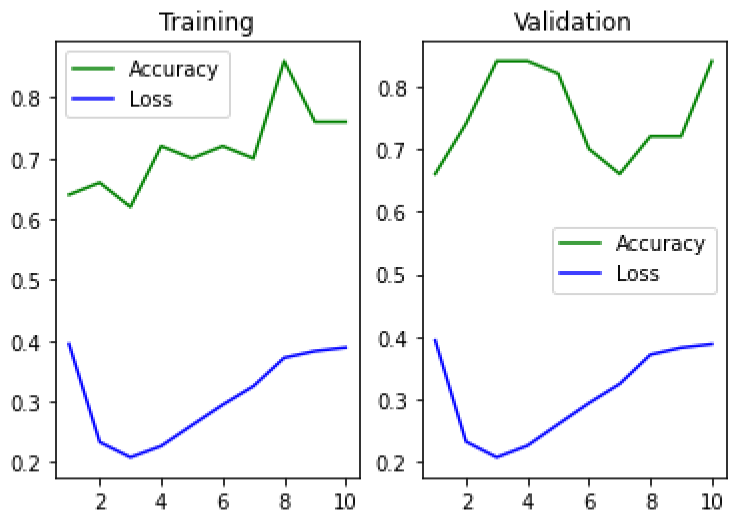
```
Count: 2, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 6701 ****************

# Epoch:=1/10  - train loss:=0.0269 - val loss:=0.0223, train acc:=0.4
4 - val acc:=0.46
# Epoch:=2/10  - train loss:=0.0210 - val loss:=0.0172, train acc:=0.6
6 - val acc:=0.76
# Epoch:=3/10  - train loss:=0.0158 - val loss:=0.0135, train acc:=0.7
2 - val acc:=0.72
# Epoch:=4/10  - train loss:=0.0121 - val loss:=0.0125, train acc:=0.8
0 - val acc:=0.68
# Epoch:=5/10  - train loss:=0.0118 - val loss:=0.0122, train acc:=0.7
2 - val acc:=0.74
# Epoch:=6/10  - train loss:=0.0143 - val loss:=0.0139, train acc:=0.7
2 - val acc:=0.80
# Epoch:=7/10  - train loss:=0.0173 - val loss:=0.0235, train acc:=0.7
6 - val acc:=0.68
# Epoch:=8/10  - train loss:=0.0171 - val loss:=0.0204, train acc:=0.7
2 - val acc:=0.76
# Epoch:=9/10  - train loss:=0.0174 - val loss:=0.0167, train acc:=0.7
6 - val acc:=0.76
# Epoch:=10/10  - train loss:=0.0189 - val loss:=0.0180, train acc:=0.
76 - val acc:=0.74

Total time taken (in seconds):  179.66

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 82.64%
*****************************************
```



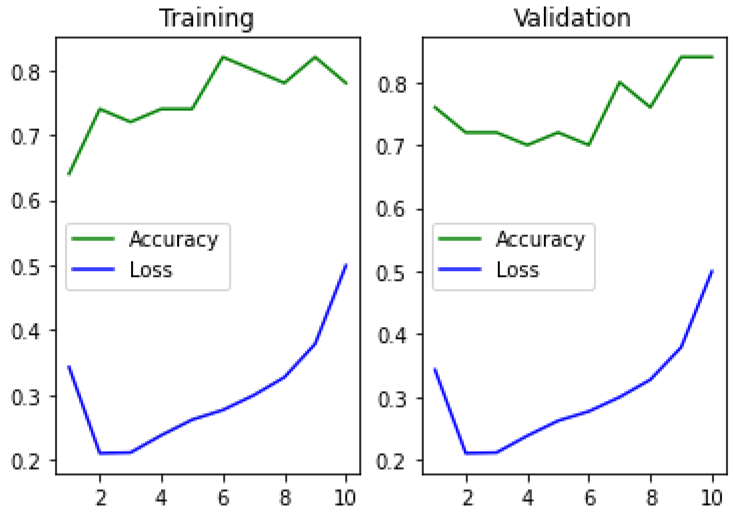
```
Count: 3, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 2904 ****************

# Epoch:=1/10  - train loss:=0.0271 - val loss:=0.0224, train acc:=0.4
4 - val acc:=0.48
# Epoch:=2/10  - train loss:=0.0219 - val loss:=0.0194, train acc:=0.5
6 - val acc:=0.48
# Epoch:=3/10  - train loss:=0.0167 - val loss:=0.0135, train acc:=0.6
8 - val acc:=0.78
# Epoch:=4/10  - train loss:=0.0123 - val loss:=0.0125, train acc:=0.8
0 - val acc:=0.78
# Epoch:=5/10  - train loss:=0.0117 - val loss:=0.0130, train acc:=0.8
2 - val acc:=0.74
# Epoch:=6/10  - train loss:=0.0139 - val loss:=0.0146, train acc:=0.7
0 - val acc:=0.74
# Epoch:=7/10  - train loss:=0.0171 - val loss:=0.0214, train acc:=0.7
6 - val acc:=0.70
# Epoch:=8/10  - train loss:=0.0218 - val loss:=0.0368, train acc:=0.7
6 - val acc:=0.76
# Epoch:=9/10  - train loss:=0.0298 - val loss:=0.0301, train acc:=0.7
6 - val acc:=0.82
# Epoch:=10/10  - train loss:=0.0337 - val loss:=0.0445, train acc:=0.
80 - val acc:=0.70

Total time taken (in seconds):  180.37

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 77.49%
*****************************************
```



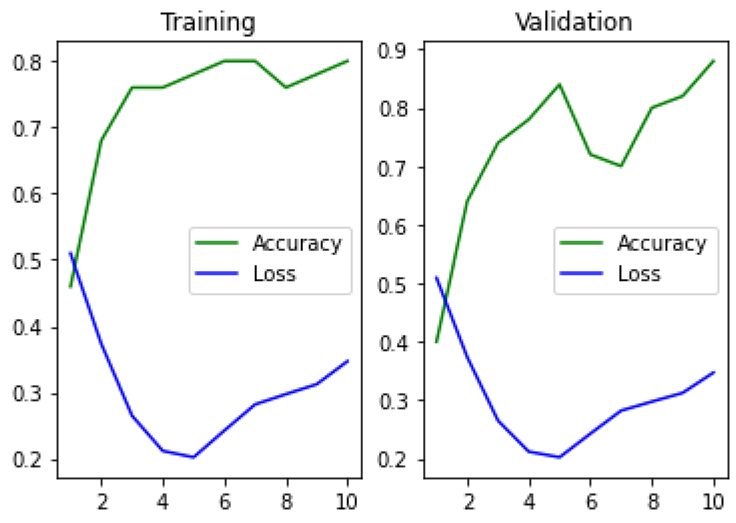
```
Count: 4, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 9218 ****************

# Epoch:=1/10  - train loss:=0.0246 - val loss:=0.0158, train acc:=0.6
4 - val acc:=0.66
# Epoch:=2/10  - train loss:=0.0145 - val loss:=0.0122, train acc:=0.6
6 - val acc:=0.74
# Epoch:=3/10  - train loss:=0.0129 - val loss:=0.0129, train acc:=0.6
2 - val acc:=0.84
# Epoch:=4/10  - train loss:=0.0141 - val loss:=0.0151, train acc:=0.7
2 - val acc:=0.84
# Epoch:=5/10  - train loss:=0.0162 - val loss:=0.0187, train acc:=0.7
0 - val acc:=0.82
# Epoch:=6/10  - train loss:=0.0183 - val loss:=0.0234, train acc:=0.7
2 - val acc:=0.70
# Epoch:=7/10  - train loss:=0.0202 - val loss:=0.0315, train acc:=0.7
0 - val acc:=0.66
# Epoch:=8/10  - train loss:=0.0231 - val loss:=0.0275, train acc:=0.8
6 - val acc:=0.72
# Epoch:=9/10  - train loss:=0.0238 - val loss:=0.0338, train acc:=0.7
6 - val acc:=0.72
# Epoch:=10/10  - train loss:=0.0242 - val loss:=0.0269, train acc:=0.
76 - val acc:=0.84

Total time taken (in seconds):  180.08

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 80.74%
*****************************************
```



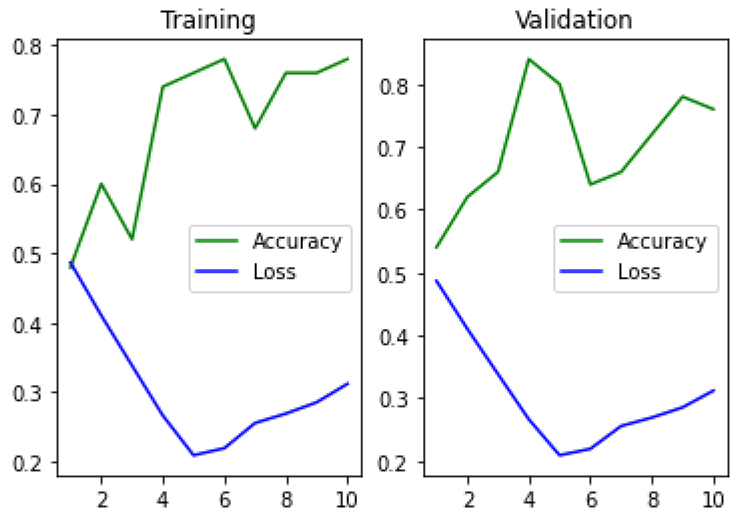
```
Count: 5, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 8514 ****************

# Epoch:=1/10  - train loss:=0.0221 - val loss:=0.0161, train acc:=0.6
4 - val acc:=0.76
# Epoch:=2/10  - train loss:=0.0136 - val loss:=0.0132, train acc:=0.7
4 - val acc:=0.72
# Epoch:=3/10  - train loss:=0.0136 - val loss:=0.0149, train acc:=0.7
2 - val acc:=0.72
# Epoch:=4/10  - train loss:=0.0153 - val loss:=0.0243, train acc:=0.7
4 - val acc:=0.70
# Epoch:=5/10  - train loss:=0.0169 - val loss:=0.0211, train acc:=0.7
4 - val acc:=0.72
# Epoch:=6/10  - train loss:=0.0178 - val loss:=0.0222, train acc:=0.8
2 - val acc:=0.70
# Epoch:=7/10  - train loss:=0.0193 - val loss:=0.0202, train acc:=0.8
0 - val acc:=0.80
# Epoch:=8/10  - train loss:=0.0211 - val loss:=0.0320, train acc:=0.7
8 - val acc:=0.76
# Epoch:=9/10  - train loss:=0.0244 - val loss:=0.0281, train acc:=0.8
2 - val acc:=0.84
# Epoch:=10/10  - train loss:=0.0322 - val loss:=0.0482, train acc:=0.
78 - val acc:=0.84

Total time taken (in seconds):  177.56

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 78.17%
*****************************************
```



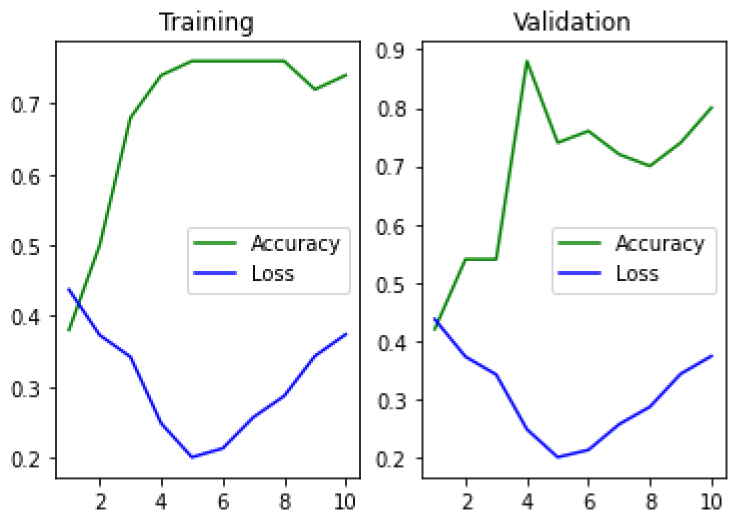
```
Count: 6, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 1087 ****************

# Epoch:=1/10  - train loss:=0.0284 - val loss:=0.0254, train acc:=0.4
6 - val acc:=0.40
# Epoch:=2/10  - train loss:=0.0208 - val loss:=0.0187, train acc:=0.6
8 - val acc:=0.64
# Epoch:=3/10  - train loss:=0.0148 - val loss:=0.0136, train acc:=0.7
6 - val acc:=0.74
# Epoch:=4/10  - train loss:=0.0118 - val loss:=0.0114, train acc:=0.7
6 - val acc:=0.78
# Epoch:=5/10  - train loss:=0.0113 - val loss:=0.0111, train acc:=0.7
8 - val acc:=0.84
# Epoch:=6/10  - train loss:=0.0135 - val loss:=0.0169, train acc:=0.8
0 - val acc:=0.72
# Epoch:=7/10  - train loss:=0.0157 - val loss:=0.0189, train acc:=0.8
0 - val acc:=0.70
# Epoch:=8/10  - train loss:=0.0166 - val loss:=0.0175, train acc:=0.7
6 - val acc:=0.80
# Epoch:=9/10  - train loss:=0.0174 - val loss:=0.0158, train acc:=0.7
8 - val acc:=0.82
# Epoch:=10/10  - train loss:=0.0194 - val loss:=0.0193, train acc:=0.
80 - val acc:=0.88

Total time taken (in seconds):  174.66

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 82.87%
*****************************************
```



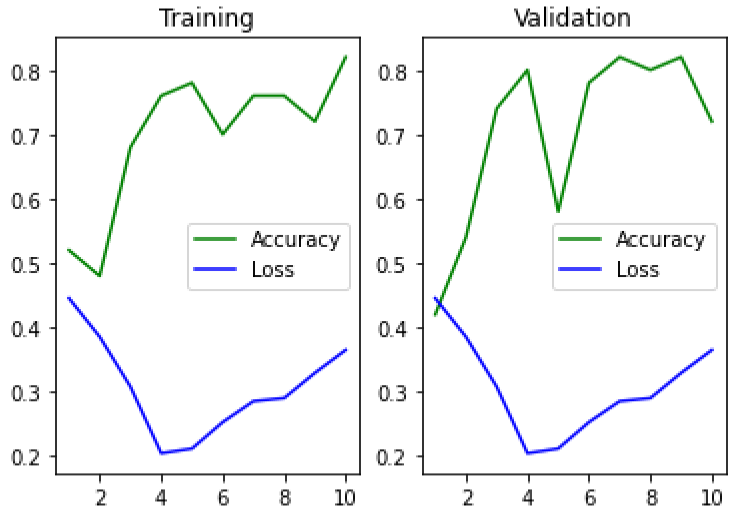
```
Count: 7, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 6826 ****************

# Epoch:=1/10  - train loss:=0.0275 - val loss:=0.0231, train acc:=0.4
8 - val acc:=0.54
# Epoch:=2/10  - train loss:=0.0232 - val loss:=0.0221, train acc:=0.6
0 - val acc:=0.62
# Epoch:=3/10  - train loss:=0.0191 - val loss:=0.0162, train acc:=0.5
2 - val acc:=0.66
# Epoch:=4/10  - train loss:=0.0151 - val loss:=0.0126, train acc:=0.7
4 - val acc:=0.84
# Epoch:=5/10  - train loss:=0.0118 - val loss:=0.0117, train acc:=0.7
6 - val acc:=0.80
# Epoch:=6/10  - train loss:=0.0124 - val loss:=0.0160, train acc:=0.7
8 - val acc:=0.64
# Epoch:=7/10  - train loss:=0.0145 - val loss:=0.0175, train acc:=0.6
8 - val acc:=0.66
# Epoch:=8/10  - train loss:=0.0152 - val loss:=0.0187, train acc:=0.7
6 - val acc:=0.72
# Epoch:=9/10  - train loss:=0.0161 - val loss:=0.0167, train acc:=0.7
6 - val acc:=0.78
# Epoch:=10/10  - train loss:=0.0176 - val loss:=0.0230, train acc:=0.
78 - val acc:=0.76

Total time taken (in seconds):  174.95

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 79.30%
*****************************************
```



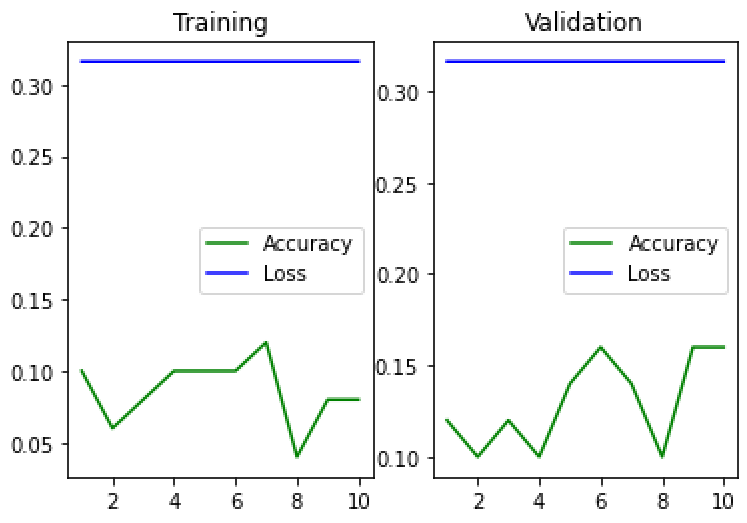
```
Count: 8, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 3699 ****************

# Epoch:=1/10  - train loss:=0.0269 - val loss:=0.0227, train acc:=0.3
8 - val acc:=0.42
# Epoch:=2/10  - train loss:=0.0229 - val loss:=0.0227, train acc:=0.5
0 - val acc:=0.54
# Epoch:=3/10  - train loss:=0.0210 - val loss:=0.0189, train acc:=0.6
8 - val acc:=0.54
# Epoch:=4/10  - train loss:=0.0152 - val loss:=0.0119, train acc:=0.7
4 - val acc:=0.88
# Epoch:=5/10  - train loss:=0.0123 - val loss:=0.0123, train acc:=0.7
6 - val acc:=0.74
# Epoch:=6/10  - train loss:=0.0131 - val loss:=0.0132, train acc:=0.7
6 - val acc:=0.76
# Epoch:=7/10  - train loss:=0.0158 - val loss:=0.0166, train acc:=0.7
6 - val acc:=0.72
# Epoch:=8/10  - train loss:=0.0176 - val loss:=0.0203, train acc:=0.7
6 - val acc:=0.70
# Epoch:=9/10  - train loss:=0.0211 - val loss:=0.0212, train acc:=0.7
2 - val acc:=0.74
# Epoch:=10/10  - train loss:=0.0230 - val loss:=0.0198, train acc:=0.
74 - val acc:=0.80

Total time taken (in seconds):  176.32

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 80.80%
*****************************************
```



```
Count: 9, j=: 0


**************** Training model: mlp_on_gpu_default with optimizer: Ad
am and seed: 3750 ****************

# Epoch:=1/10  - train loss:=0.0274 - val loss:=0.0250, train acc:=0.5
2 - val acc:=0.42
# Epoch:=2/10  - train loss:=0.0237 - val loss:=0.0221, train acc:=0.4
8 - val acc:=0.54
# Epoch:=3/10  - train loss:=0.0189 - val loss:=0.0137, train acc:=0.6
8 - val acc:=0.74
# Epoch:=4/10  - train loss:=0.0126 - val loss:=0.0114, train acc:=0.7
6 - val acc:=0.80
# Epoch:=5/10  - train loss:=0.0130 - val loss:=0.0236, train acc:=0.7
8 - val acc:=0.58
# Epoch:=6/10  - train loss:=0.0156 - val loss:=0.0151, train acc:=0.7
0 - val acc:=0.78
# Epoch:=7/10  - train loss:=0.0176 - val loss:=0.0185, train acc:=0.7
6 - val acc:=0.82
# Epoch:=8/10  - train loss:=0.0179 - val loss:=0.0241, train acc:=0.7
6 - val acc:=0.80
# Epoch:=9/10  - train loss:=0.0203 - val loss:=0.0202, train acc:=0.7
2 - val acc:=0.82
# Epoch:=10/10  - train loss:=0.0225 - val loss:=0.0290, train acc:=0.
82 - val acc:=0.72

Total time taken (in seconds):  174.45

Finished training model: mlp_on_gpu_default

**************** Testing ****************
mlp_on_gpu_default model accuracy = 78.34%
*****************************************
```



```
Count: 0, j=: 0


**************** Training model: mlp_on_gpu_RegL1 with optimizer: Adam
and seed: 6000 ****************

# Epoch:=1/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.1
0 - val acc:=0.12
# Epoch:=2/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.0
6 - val acc:=0.10
# Epoch:=3/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.0
8 - val acc:=0.12
# Epoch:=4/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.1
0 - val acc:=0.10
# Epoch:=5/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.1
0 - val acc:=0.14
# Epoch:=6/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.1
0 - val acc:=0.16
# Epoch:=7/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.1
2 - val acc:=0.14
# Epoch:=8/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.0
4 - val acc:=0.10
# Epoch:=9/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.0
8 - val acc:=0.16
# Epoch:=10/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.
08 - val acc:=0.16

Total time taken (in seconds):  188.25

Finished training model: mlp_on_gpu_RegL1

**************** Testing ****************
mlp_on_gpu_RegL1 model accuracy = 10.00%
*****************************************
```



```
Count: 1, j=: 0


**************** Training model: mlp_on_gpu_RegL1 with optimizer: Adam
and seed: 4161 ****************

# Epoch:=1/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.0
6 - val acc:=0.12
# Epoch:=2/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.1
2 - val acc:=0.08
# Epoch:=3/10  - train loss:=0.0461 - val loss:=0.0461, train acc:=0.0
6 - val acc:=0.06
```