COMP0090 Coursework 3

Tom Grigg
19151291
tom.grigg.19@ucl.ac.uk

Oliver Slumbers
19027699
oliver.slumbers.19@ucl.ac.uk

Agnieszka Dobrowolska
16034489
zcqsaad@ucl.ac.uk

Charita Dellaporta
19025680
charita.dellaporta.19@ucl.ac.uk

Sibo Zhao
19099708
ucabsz6@ucl.ac.uk

January 2020

1 Hyperlinks

Task 1: `https://drive.google.com/open?id=1uFeP-ZxRgCTfCLMZUy0hanJF87nvb0bI`

Task 2: `https://drive.google.com/open?id=1bos8MBqtLjuYBOU2D9bpv6WYFhgENCf9`

Task 3: `https://drive.google.com/open?id=163SD5E0t9-6jwv9tvADjt6DyQgexXTq1`

Task 4: `https://drive.google.com/open?id=1kMgpMzQCHwSYYfguEHaCF8-syvXYLlSx`

Task 5: `https://drive.google.com/open?id=13xkLb8_mnnPgWYe6M8thcrQY2EHdap3u`

2 Contributions inside the group

Harita, Sibo and Agnieszka worked on Q1,2 and 3, while Tom and Oliver and Tom completed questions 4 and 5.

Q1

Question 1

In [0]:
```
import os
import gzip
import numpy as np
os.system("pip install python-mnist")
from mnist import MNIST
import matplotlib.pyplot as plt
%matplotlib inline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.autograd import Variable
from sklearn.metrics import confusion_matrix
import pandas as pd
```

In [0]:
```
# Download the dataset.
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-images-idx3-ubyte.gz -o /tmp/train-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-labels-idx1-ubyte.gz -o /tmp/train-labels-idx1-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-images-idx3-ubyte.gz -o /tmp/t10k-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-labels-idx1-ubyte.gz -o /tmp/t10k-labels-idx1-ubyte.gz')
pass
```

In [0]:
```
# load the dataset with shape (*, 784)
mnistdata = MNIST("/tmp")
mnistdata.gz = True
trainxs_raw, trainys_raw = mnistdata.load_training()
testxs_raw, testys_raw = mnistdata.load_testing()

# reshape data into square image (*, 1, 28, 28)
trainxs_square  = np.array(trainxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
trainys_square  = np.array(trainys_raw, dtype=np.long)
testxs          = np.array(testxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
testys          = np.array(testys_raw, dtype=np.long)

# split training dataset
trainxs = trainxs_square[:50000]
trainys = trainys_square[:50000]
validxs = trainxs_square[50000:]
validys = trainys_square[50000:]
```

In [0]:
```
# process into FashionMNist 1 and Fashon MNist 2

def filter_for_labels(trainxs, trainys, validxs, validys, testxs, testys, labels):
    # FashinoMNist 1
    train_ix = np.isin(trainys, labels)
    trainxs_ = trainxs[train_ix]
    trainys_ = trainys[train_ix]

    val_ix = np.isin(validys, labels)
    validxs_ = validxs[val_ix]
    validys_ = validys[val_ix]

    test_ix = np.isin(testys, labels)
    testxs_ = testxs[test_ix]
    testys_ = testys[test_ix]

    return trainxs_, trainys_, validxs_, validys_, testxs_, testys_

labels_1 = [0, 1, 4, 5, 8]
labels_2 = [x for x in list(range(10)) if x not in labels_1]

# FashinoMNist1
trainxs_1, trainys_1, validxs_1, validys_1, testxs_1, testys_1 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_1
)

def change_labels(ys, labels):
  new_ys = np.zeros(len(ys))
  position_dict = {labels[i]: i for i in range(len(labels))}
  for label in position_dict.keys():
    new_ys[ys == label] = position_dict[label]
  return new_ys

trainys_1 = change_labels(trainys_1, labels_1)
validys_1 = change_labels(validys_1, labels_1)
testys_1 = change_labels(testys_1, labels_1)

trainys_1 = np.array(trainys_1, dtype=np.long)
validys_1 = np.array(validys_1, dtype=np.long)
testys_1 = np.array(testys_1, dtype=np.long)

#FashionMNist2
trainxs_2, trainys_2, validxs_2, validys_2, testxs_2, testys_2 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_2
)
```

In [0]:
```
class DatasetConstructor(Dataset):

    def __init__(self, X, Y):
        self.X = torch.from_numpy(X)
        self.Y = torch.from_numpy(Y)
        self.len = X.shape[0]

    def __getitem__(self, index):
        return self.X[index], self.Y[index]

    def __len__(self):
        return self.len

BATCHSIZE = 32

train_data = DatasetConstructor(trainxs_1, trainys_1)
valid_data = DatasetConstructor(validxs_1, validys_1)
test_data  = DatasetConstructor(testxs_1, testys_1)


train_loader  = DataLoader(dataset=train_data, batch_size=BATCHSIZE,
                           shuffle = True, num_workers=2)
valid_loader  = DataLoader(dataset=valid_data, batch_size=BATCHSIZE,
                           shuffle = True, num_workers=2)
test_loader   = DataLoader(dataset=test_data, batch_size=BATCHSIZE,
                           shuffle = True, num_workers=2)

# tensor dataset for loss and accuracy computation
trainxs_tensor = torch.from_numpy(trainxs_1)
trainys_tensor = torch.from_numpy(trainys_1)
validxs_tensor = torch.from_numpy(validxs_1)
validys_tensor = torch.from_numpy(validys_1)
testxs_tensor  = torch.from_numpy(testxs_1)
testys_tensor  = torch.from_numpy(testys_1)
```

In [0]:
```
# dataset examples
fig=plt.figure(figsize=(20, 5))
columns = 20
rows = 5
for i in range(columns*rows):
    fig.add_subplot(rows, columns, i+1)
    plt.axis('off')
    plt.imshow(trainxs_1[i, 0], cmap='binary')
plt.show()
```

1. Implement a multi-class, convolutional neural network with cross-entropy loss for the Fashion-MNIST-1 data.

In [0]:
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        #data size 1*28*28 -> 4*14*14
        self.cnn_layer1 = nn.Sequential(
            nn.Conv2d(1, 4, 3, padding=1, padding_mode="same"),
            nn.BatchNorm2d(4),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2)
        )
        #data size 4*14*14 -> 4*7*7
        self.cnn_layer2 = nn.Sequential(
            nn.Conv2d(4, 4, 3, padding=1, padding_mode="same"),
            nn.BatchNorm2d(4),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2)
        )
        self.linear_layers = nn.Sequential(
            nn.Linear(4 * 7 * 7, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, 5)
        )

    def forward(self, x):

        x = self.cnn_layer1(x)

        x = self.cnn_layer2(x)
        x = x.view(x.size(0), -1)# reshape
        x = self.linear_layers(x)
        return x

def accuracy(model, X, Y):
    outputs = model(X)
    _, prediction = torch.max(outputs, 1)
    if torch.cuda.is_available():
        prediction = prediction.cpu()
    return sum((Y - prediction.numpy())==0) / len(Y) * 100
```

2. Train your final model to convergence on the training set using an optimisation algorithm of your choice.

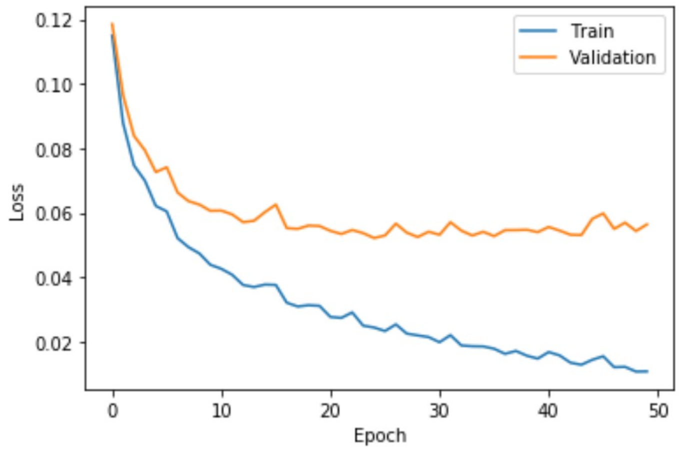
In [0]:
```
model = CNN()
if torch.cuda.is_available():
    model.cuda()

loss = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
#optimizer = optim.Adam(model.parameters(), lr=0.05)

train_loss = []
valid_loss = []
epochs = 50

for epoch in range(epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        if torch.cuda.is_available():
            inputs, labels = inputs.cuda(), labels.cuda()
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss_val = loss(outputs, labels)
        loss_val.backward()
        optimizer.step()

        # print statistics
        running_loss += loss_val.item()
        if i % 200 == 199:    # print every 200 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, (i + 1)*BATCHSIZE, running_loss / 200))
            running_loss = 0.0

    if torch.cuda.is_available():
        train_loss.append(loss(model(trainxs_tensor.cuda()), trainys_tensor.cuda
()).item())
        valid_loss.append(loss(model(validxs_tensor.cuda()), validys_tensor.cuda
()).item())
    else:
        train_loss.append(loss(model(trainxs_tensor), trainys_tensor).item())
        valid_loss.append(loss(model(validxs_tensor), validys_tensor).item())

print('Finished Training')

if torch.cuda.is_available():
    print("Training data accuracy:  ", accuracy(model, trainxs_tensor.cuda(), train
ys_1))
    print("Validation data accuracy:", accuracy(model, validxs_tensor.cuda(), valid
ys_1))
    print("Testing data accuracy:   ", accuracy(model, testxs_tensor.cuda(), testys
_1))
    accuracy_train = accuracy(model, trainxs_tensor.cuda(), trainys_1)
    accuracy_valid = accuracy(model, validxs_tensor.cuda(), validys_1)
    accuracy_test = accuracy(model, testxs_tensor.cuda(), testys_1)
else:
    print("Training data accuracy:  ", accuracy(model, trainxs_tensor, trainys_1))
    print("Validation data accuracy:", accuracy(model, validxs_tensor, validys_1))
    print("Testing data accuracy:   ", accuracy(model, testxs_tensor, testys_1))
    accuracy_train = accuracy(model, trainxs_tensor, trainys_1)
    accuracy_valid = accuracy(model, validxs_tensor, validys_1)
```

```
[1,  6400] loss: 0.732
[1, 12800] loss: 0.193
[1, 19200] loss: 0.135
[2,  6400] loss: 0.103
[2, 12800] loss: 0.102
[2, 19200] loss: 0.098
[3,  6400] loss: 0.079
[3, 12800] loss: 0.089
[3, 19200] loss: 0.082
[4,  6400] loss: 0.080
[4, 12800] loss: 0.072
[4, 19200] loss: 0.077
[5,  6400] loss: 0.063
[5, 12800] loss: 0.069
[5, 19200] loss: 0.072
[6,  6400] loss: 0.055
[6, 12800] loss: 0.068
[6, 19200] loss: 0.060
[7,  6400] loss: 0.058
[7, 12800] loss: 0.056
[7, 19200] loss: 0.057
[8,  6400] loss: 0.056
[8, 12800] loss: 0.057
[8, 19200] loss: 0.053
[9,  6400] loss: 0.045
[9, 12800] loss: 0.056
[9, 19200] loss: 0.044
[10,  6400] loss: 0.051
[10, 12800] loss: 0.055
[10, 19200] loss: 0.043
[11,  6400] loss: 0.039
[11, 12800] loss: 0.044
[11, 19200] loss: 0.052
[12,  6400] loss: 0.034
[12, 12800] loss: 0.044
[12, 19200] loss: 0.046
[13,  6400] loss: 0.035
[13, 12800] loss: 0.046
[13, 19200] loss: 0.042
[14,  6400] loss: 0.037
[14, 12800] loss: 0.046
[14, 19200] loss: 0.038
[15,  6400] loss: 0.045
[15, 12800] loss: 0.038
[15, 19200] loss: 0.036
[16,  6400] loss: 0.043
[16, 12800] loss: 0.033
[16, 19200] loss: 0.037
[17,  6400] loss: 0.030
[17, 12800] loss: 0.039
[17, 19200] loss: 0.036
[18,  6400] loss: 0.037
[18, 12800] loss: 0.031
[18, 19200] loss: 0.036
[19,  6400] loss: 0.029
[19, 12800] loss: 0.032
[19, 19200] loss: 0.041
[20,  6400] loss: 0.031
[20, 12800] loss: 0.032
[20, 19200] loss: 0.036
[21,  6400] loss: 0.035
[21, 12800] loss: 0.029
[21, 19200] loss: 0.030
[22,  6400] loss: 0.036
```

3. Provide a plot of the loss on the training set and validation set for each epoch of training.

In [0]:
```
fig = plt.figure()
plt.plot(train_loss)
plt.plot(valid_loss)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(['Train', 'Validation'])
```

Out[0]:
```
<matplotlib.legend.Legend at 0x7fa725c1ce10>
```



4. Provide the final accuracy on the training, validation, and test set.

In [0]:
```
print('Training accuracy is {} %.'.format(accuracy_train))
print('Validation accuracy is {} %.'.format(accuracy_valid))
print('Test accuracy is {} %'.format(accuracy_test))
```

```
Training accuracy is 99.76776776776777 %.
Validation accuracy is 98.42786069651741 %.
Test accuracy is 98.66 %
```

5. Analyse the errors of your models by constructing a confusion matrix. Which classes are easily “confused” by the model? Hypothesise why.

In [0]:
```
# Confusion matrix
temp_model = CNN()
temp_model.load_state_dict(torch.load(PATH))
outputs = temp_model(torch.from_numpy(testxs_1))
_, prediction = torch.max(outputs, 1)
cm = confusion_matrix(testys_1,prediction.numpy())

label_names = ['Tshirt', 'Trousr',  'Coat',
               'Sandal', 'Bag']
confusion_df = pd.DataFrame(data=cm[0:,0:], index=label_names, columns=label_names)
display(confusion_df)
```

	Tshirt	Trousr	Coat	Sandal	Bag
Tshirt	978	2	11	0	9
Trousr	5	989	3	1	2
Coat	7	2	987	0	4
Sandal	0	0	0	997	3
Bag	7	1	4	6	982

In [0]:
```
# Print one piece of clothing from each label
fig=plt.figure(figsize=(20, 5))
columns = 5
rows = 1
for i in range(5):
  fig.add_subplot(rows, columns, i+1)
  plt.axis('off')
  index = np.nonzero(testys_1 == np.unique(testys_1)[i])[0][0]
  plt.imshow(testxs_1[index,0], cmap='binary')
plt.show()
```

Above we plot one piece of clothing from each label to be able to analyse the confusion matrix. As can be seen in the above confusion matrix, similarly looking items like t-shirts and coats are most often confused with each other. We also observe that bags are often confused with tshirts and coats - this is likely because all them are angular and have sharp, straight vertical edges. Overall, we find that sandals are the items that are easiest to predict in this set. This is something we would expect as the features of a sandal are most distinct in comparison with all the other items of clothing.

In [0]:

Q2

Question 2

In [0]:
```
import os
import gzip
import numpy as np
os.system("pip install python-mnist")
from mnist import MNIST
import matplotlib.pyplot as plt
%matplotlib inline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.autograd import Variable
from sklearn.metrics import confusion_matrix
import pandas as pd
from tqdm import tqdm
```

In [0]:
```
# Download the dataset.
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-images-idx3-ubyte.gz -o /tmp/train-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-labels-idx1-ubyte.gz -o /tmp/train-labels-idx1-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-images-idx3-ubyte.gz -o /tmp/t10k-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-labels-idx1-ubyte.gz -o /tmp/t10k-labels-idx1-ubyte.gz')
pass
```

In [0]:
```
# load the dataset with shape (*, 784)
mnistdata = MNIST("/tmp")
mnistdata.gz = True
trainxs_raw, trainys_raw = mnistdata.load_training()
testxs_raw, testys_raw = mnistdata.load_testing()

# reshape data into square image (*, 1, 28, 28)
trainxs_square  = np.array(trainxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
trainys_square  = np.array(trainys_raw, dtype=np.long)
testxs          = np.array(testxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
testys          = np.array(testys_raw, dtype=np.long)

# split training dataset
trainxs = trainxs_square[:50000]
trainys = trainys_square[:50000]
validxs = trainxs_square[50000:]
validys = trainys_square[50000:]
```

In [0]:
```
# process into FashionMNist 1 and Fashon MNist 2

def filter_for_labels(trainxs, trainys, validxs, validys, testxs, testys, labels):
    # FashinoMNist 1
    train_ix = np.isin(trainys, labels)
    trainxs_ = trainxs[train_ix]
    trainys_ = trainys[train_ix]

    val_ix = np.isin(validys, labels)
    validxs_ = validxs[val_ix]
    validys_ = validys[val_ix]

    test_ix = np.isin(testys, labels)
    testxs_ = testxs[test_ix]
    testys_ = testys[test_ix]

    return trainxs_, trainys_, validxs_, validys_, testxs_, testys_

labels_1 = [0, 1, 4, 5, 8]
labels_2 = [x for x in list(range(10)) if x not in labels_1]

# FashinoMNist1
trainxs_1, trainys_1, validxs_1, validys_1, testxs_1, testys_1 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_1
)

def change_labels(ys, labels):
  new_ys = np.zeros(len(ys))
  position_dict = {labels[i]: i for i in range(len(labels))}
  for label in position_dict.keys():
    new_ys[ys == label] = position_dict[label]
  return new_ys

trainys_1 = change_labels(trainys_1, labels_1)
validys_1 = change_labels(validys_1, labels_1)
testys_1 = change_labels(testys_1, labels_1)

trainys_1 = np.array(trainys_1, dtype=np.long)
validys_1 = np.array(validys_1, dtype=np.long)
testys_1 = np.array(testys_1, dtype=np.long)

#FashionMNist2
trainxs_2, trainys_2, validxs_2, validys_2, testxs_2, testys_2 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_2
)
```

In [0]:
```
class DatasetConstructor(Dataset):

    def __init__(self, X, Y):
        self.X = torch.from_numpy(X)
        self.Y = torch.from_numpy(Y)
        self.len = X.shape[0]

    def __getitem__(self, index):
        return self.X[index], self.Y[index]

    def __len__(self):
        return self.len

BATCHSIZE = 32

train_data = DatasetConstructor(trainxs_1, trainys_1)
valid_data = DatasetConstructor(validxs_1, validys_1)
test_data  = DatasetConstructor(testxs_1, testys_1)

train_loader  = DataLoader(dataset=train_data, batch_size=BATCHSIZE,
                           shuffle = True, num_workers=2)
valid_loader  = DataLoader(dataset=valid_data, batch_size=BATCHSIZE,
                           shuffle = True, num_workers=2)
test_loader   = DataLoader(dataset=test_data, batch_size=BATCHSIZE,
                           shuffle = True, num_workers=2)

# tensor dataset for loss and accuracy computation
trainxs_tensor = torch.from_numpy(trainxs_1)
trainys_tensor = torch.from_numpy(trainys_1)
validxs_tensor = torch.from_numpy(validxs_1)
validys_tensor = torch.from_numpy(validys_1)
testxs_tensor  = torch.from_numpy(testxs_1)
testys_tensor  = torch.from_numpy(testys_1)
```

In [0]:
```
# dataset examples
fig=plt.figure(figsize=(20, 5))
columns = 20
rows = 5
for i in range(columns*rows):
    fig.add_subplot(rows, columns, i+1)
    plt.axis('off')
    plt.imshow(trainxs_1[i, 0], cmap='binary')
plt.show()
```

1. Implement a multi-class, convolutional neural network with cross-entropy loss for the Fashion-MNIST-1 data, then fill in information regarding it into a table akin to Table 1.6

In [0]:
```
# Class to define model of arbitrarily sized architecture

class CNN_class(nn.Module):

    def __init__(self, input_size, channels_conv, pool_size, kernel_size, hids, out
put_size):
        super(CNN_class, self).__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.channels_conv = channels_conv
        self.kernel_size = kernel_size
        self.pool_size = pool_size
        self.hids = hids
        self.conv_layers = []
        self.classification_layers = []

        assert len(channels_conv) > 0
        assert len(hids) > 0
        assert len(channels_conv) == len(kernel_size) == len(pool_size)
        assert all(i > 0 for i in hids)
        assert all(i > 0 for i in channels_conv)
        assert all(i > 0 for i in kernel_size)

        self.conv_layers.append(nn.Conv2d(1, self.channels_conv[0], self.kernel_siz
e[0], padding=1, padding_mode="same"))
        self.conv_layers.append(nn.BatchNorm2d(self.channels_conv[0]))
        self.conv_layers.append(nn.ReLU(inplace=True))
        self.conv_layers.append(nn.MaxPool2d(self.pool_size[0],self.pool_size[0]))

        for i in range(1,len(self.channels_conv)):
          self.conv_layers.append(nn.Conv2d(self.channels_conv[i-1], self.channels_
conv[i], self.kernel_size[i], padding=1, padding_mode="same"))
          self.conv_layers.append(nn.BatchNorm2d(self.channels_conv[i]))
          self.conv_layers.append(nn.ReLU(inplace=True))
          self.conv_layers.append(nn.MaxPool2d(self.pool_size[i],self.pool_size
[i]))

        self.conv = nn.Sequential(*self.conv_layers)

        p = np.array(self.input_size) // self.pool_size[0][0]
        for i in range(1,len(self.channels_conv)):
          p = p // self.pool_size[i][0]

        self.conv_out_size = int(p[0]*p[1]*self.channels_conv[-1])

        self.classification_layers.append(nn.Linear(self.conv_out_size, self.hids
[0]))
        self.classification_layers.append(nn.ReLU(inplace=True))

        for i in range(1,len(hids)):
          self.classification_layers.append(nn.Linear(self.hids[i-1], self.hids
[i]))
          self.classification_layers.append(nn.ReLU(inplace=True))

        self.classification_layers.append(nn.Linear(self.hids[-1], self.output_siz
e))

        self.classif = nn.Sequential(*self.classification_layers)

    def forward(self, x):

      x = self.conv(x)
      x = x.view(x.size(0), -1)      # reshape
      x = self.classif(x)
```

In [0]:
```
# Check that you get the model you expect

input_size = [28,28]
channels_conv = [15,15,15]
pool_size = [[2,2], [2,2], [2,2]]
kernel_size = [3,3,3]
hids = [128]
output_size = 5
lr = 0.001
reg = 2
lmbd = 0.001
num_of_epochs = 20
opt = 'SGD'

model = CNN_class(input_size, channels_conv, pool_size, kernel_size, hids, output_s
ize)
model
```

Out[0]:
```
CNN_class(
  (conv): Sequential(
    (0): Conv2d(1, 15, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), paddin
g_mode=same)
    (1): BatchNorm2d(15, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=[2, 2], stride=[2, 2], padding=0, dilation=1, cei
l_mode=False)
    (4): Conv2d(15, 15, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), paddi
ng_mode=same)
    (5): BatchNorm2d(15, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=[2, 2], stride=[2, 2], padding=0, dilation=1, cei
l_mode=False)
    (8): Conv2d(15, 15, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), paddi
ng_mode=same)
    (9): BatchNorm2d(15, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    (10): ReLU(inplace=True)
    (11): MaxPool2d(kernel_size=[2, 2], stride=[2, 2], padding=0, dilation=1, ce
il_mode=False)
  )
  (classif): Sequential(
    (0): Linear(in_features=135, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=5, bias=True)
  )
)
```

In [0]:
```
def accuracy(model, X, Y):
    outputs = model(X)
    _, prediction = torch.max(outputs, 1)
    if torch.cuda.is_available():
        prediction = prediction.cpu()
    return sum((Y - prediction.numpy())==0) / len(Y) * 100
```

In [0]:
```
# Training function for input training parameters

def train_CNN(model, lr, reg, lambd, num_of_epochs, opt=''):

  # Generic function to train model with input training parameters:
  # lr = learning rate
  # reg = 1 or 2 for L1 or L2 regularisation
  # lambd = parameter lambda of regularisation - set to lambda = 0 for no regularis
ation
  # num_of_epochs = number of epochs for training loop
  # opt = keyword argument set to SGD or ADAM

  if torch.cuda.is_available():
    model.cuda()

  loss = nn.CrossEntropyLoss()
  if opt == 'SGD':
    optimizer = optim.SGD(model.parameters(), lr=lr)
  else:
    optimizer = optim.Adam(model.parameters(), lr=lr)

  train_loss = []
  valid_loss = []
  test_loss = []

  for epoch in tqdm(range(num_of_epochs)):

    running_loss = 0.0
    running_loss_without_reg = 0.0
    for i, data in enumerate(train_loader, 0):
      inputs, labels = data

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = model(inputs)
      loss_val_pre = loss(outputs, labels)
      regularisation = 0
      for param in model.parameters():
          regularisation += torch.norm(param, reg)
      loss_val = loss_val_pre + lambd*regularisation
      loss_val.backward()
      optimizer.step()

      # print statistics
      #running_loss += loss_val.item()
      #running_loss_without_reg += loss_val_pre.item()
      #if i % 200 == 199:    # print every 200 mini-batches
          #print('with reg [%d, %5d] loss: %.3f' %
                #(epoch + 1, (i + 1)*BATCHSIZE, running_loss / 200))
          #running_loss = 0.0
          #print('without reg [%d, %5d] loss: %.3f' %
                #(epoch + 1, (i + 1)*BATCHSIZE, running_loss_without_reg / 200))
          #running_loss_without_reg = 0.0

    if torch.cuda.is_available():
        train_loss.append(loss(model(trainxs_tensor.cuda()), trainys_tensor.cuda
()).item())
        valid_loss.append(loss(model(validxs_tensor.cuda()), validys_tensor.cuda
()).item())
```

In [0]:

```
# Function to train and append results to a data frame
def train_and_append(df, input_size, channels_conv, pool_size, kernel_size, hids, output_size, lr, reg, lmbd, num_of_epochs, opt):


  model = CNN_class(input_size, channels_conv, pool_size, kernel_size, hids, output_size)
  train_loss, valid_loss, test_loss, final_accuracies = train_CNN(model, lr, reg, lmbd, num_of_epochs, opt=opt)

  df = df.append({'Train_accuracy (%)': final_accuracies[0], 'Valid_accuracy (%)': final_accuracies[1], 'Test_accuracy (%)': final_accuracies[2],
                  'Train_Loss': train_loss[-1], 'Valid_Loss': valid_loss[-1], 'Test_Loss': test_loss[-1],
                  'Convolution layers': channels_conv, 'Classification hidden units': hids, 'Optimiser': str(opt), 'Learning rate': lr,
                  'Regularisation/ parameter (lambda)': "L {}, lambda = {}".format(reg, lmbd)}, ignore_index=True)

  return df
```

In [0]:

```
# Create an initial data frame
data_frame = {'Convolution layers':[],
        'Classification hidden units':[],
        'Optimiser':[], 'Learning rate':[], 'Regularisation/ parameter (lambda)':[],
        'Train_Loss':[], 'Valid_Loss':[], 'Test_Loss':[],
        'Train_accuracy (%)':[], 'Valid_accuracy (%)':[], 'Test_accuracy (%)':[]
       }
df = pd.DataFrame(data_frame)

# Initial model
df = train_and_append(df, [28,28], [4,4], [[2,2],[2,2]], [3,3], [128], 5, 0.001, 1, 0, 20, 'SGD') # Initial model 1
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:17<05:37, 17.78s/it]
 10%|█         | 2/20 [00:34<05:15, 17.55s/it]
 15%|█▌        | 3/20 [00:51<04:53, 17.26s/it]
 20%|██        | 4/20 [01:07<04:31, 16.94s/it]
 25%|██▌       | 5/20 [01:23<04:10, 16.70s/it]
 30%|███       | 6/20 [01:39<03:51, 16.55s/it]
 35%|███▌      | 7/20 [01:56<03:34, 16.51s/it]
 40%|████      | 8/20 [02:12<03:16, 16.38s/it]
 45%|████▌     | 9/20 [02:28<02:58, 16.22s/it]
 50%|█████     | 10/20 [02:44<02:42, 16.23s/it]
 55%|█████▌    | 11/20 [03:00<02:26, 16.30s/it]
 60%|██████    | 12/20 [03:19<02:14, 16.84s/it]
 65%|██████▌   | 13/20 [03:35<01:56, 16.64s/it]
 70%|███████   | 14/20 [03:51<01:39, 16.62s/it]
 75%|███████▌  | 15/20 [04:09<01:24, 16.87s/it]
 80%|████████  | 16/20 [04:25<01:06, 16.62s/it]
 85%|████████▌ | 17/20 [04:41<00:49, 16.49s/it]
 90%|█████████ | 18/20 [04:57<00:32, 16.47s/it]
 95%|█████████▌| 19/20 [05:14<00:16, 16.46s/it]
100%|██████████| 20/20 [05:30<00:00, 16.43s/it]
```

```
In [0]: display(df)
```

	Convolution layers	Classification hidden units	Optimiser	Learning rate	Regularisation/ parameter (lambda)	Train_Loss	Valid_Loss	Test_Loss	Train_a
0	[4, 4]	[128]	SGD	0.001	L 1, lambda = 0	0.066362	0.079999	0.07329	9ε

2. Iteratively make modifications to your model based on how your changes affect the validation loss, try to minimise it by producing nine additional variants. Note that you will need to construct one loss function without regularisation and one with regularisation, the former to obtain the loss to enter into your table and the latter to obtain your gradients, or your losses will not be comparable as you change the regulariser. It is also a good idea to plot the training and validation loss across epochs, rather than simply observing the final validation loss as the shape of the curves provide further insights into the model performance.

In [0]:
```
# Train and plot to see results - we use this to experiment with different models a
nd observe the final validation loss
# as well as the plot of the validation and the training losses across epochs.

input_size = [28,28]
channels_conv = [20,20,20]
pool_size = [[2,2], [2,2], [2,2]]
kernel_size = [3,3,3]
hids = [128]
output_size = 5
lr = 0.0001
reg = 2
lmbd = 0.01
num_of_epochs = 20
opt = 'ADAM'

model = CNN_class(input_size, channels_conv, pool_size, kernel_size, hids, output_s
ize)
train_loss, valid_loss, test_loss, final_accuracies = train_CNN(model, lr, reg, lmb
d, num_of_epochs, opt=opt)

print("Final validation loss:", valid_loss[-1])   # check final validation loss

fig = plt.figure()      # plot training and validation loss for each epoch
plt.plot(train_loss)
plt.plot(valid_loss)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(['Train', 'Validation'])
```

```
  0%|          | 0/20 [00:00<?, ?it/s]

with reg [1,  3200] loss: 1.230
without reg [1,  3200] loss: 0.923
with reg [1,  6400] loss: 0.600
without reg [1,  6400] loss: 0.294
with reg [1,  9600] loss: 0.492
without reg [1,  9600] loss: 0.187
with reg [1, 12800] loss: 0.446
without reg [1, 12800] loss: 0.143
with reg [1, 16000] loss: 0.417
without reg [1, 16000] loss: 0.116
with reg [1, 19200] loss: 0.401
without reg [1, 19200] loss: 0.100
with reg [1, 22400] loss: 0.393
without reg [1, 22400] loss: 0.095

  5%|▌         | 1/20 [00:56<17:47, 56.18s/it]

with reg [2,  3200] loss: 0.371
without reg [2,  3200] loss: 0.076
with reg [2,  6400] loss: 0.360
without reg [2,  6400] loss: 0.066
with reg [2,  9600] loss: 0.368
without reg [2,  9600] loss: 0.077
with reg [2, 12800] loss: 0.356
without reg [2, 12800] loss: 0.066
with reg [2, 16000] loss: 0.359
without reg [2, 16000] loss: 0.071
with reg [2, 19200] loss: 0.356
without reg [2, 19200] loss: 0.069
with reg [2, 22400] loss: 0.350
without reg [2, 22400] loss: 0.064

 10%|█         | 2/20 [01:48<16:31, 55.10s/it]

with reg [3,  3200] loss: 0.342
without reg [3,  3200] loss: 0.058
with reg [3,  6400] loss: 0.339
without reg [3,  6400] loss: 0.056
with reg [3,  9600] loss: 0.348
without reg [3,  9600] loss: 0.066
with reg [3, 12800] loss: 0.326
without reg [3, 12800] loss: 0.045
with reg [3, 16000] loss: 0.332
without reg [3, 16000] loss: 0.052
with reg [3, 19200] loss: 0.323
without reg [3, 19200] loss: 0.044
with reg [3, 22400] loss: 0.328
without reg [3, 22400] loss: 0.049

 15%|█▌        | 3/20 [02:41<15:26, 54.47s/it]
```

```
with reg [4,  3200] loss: 0.323
without reg [4,  3200] loss: 0.046
with reg [4,  6400] loss: 0.318
without reg [4,  6400] loss: 0.042
with reg [4,  9600] loss: 0.318
without reg [4,  9600] loss: 0.043
with reg [4, 12800] loss: 0.310
without reg [4, 12800] loss: 0.036
with reg [4, 16000] loss: 0.324
without reg [4, 16000] loss: 0.050
with reg [4, 19200] loss: 0.316
without reg [4, 19200] loss: 0.043
with reg [4, 22400] loss: 0.317
without reg [4, 22400] loss: 0.045

 20%|██        | 4/20 [03:34<14:21, 53.86s/it]

with reg [5,  3200] loss: 0.310
without reg [5,  3200] loss: 0.040
with reg [5,  6400] loss: 0.308
without reg [5,  6400] loss: 0.039
with reg [5,  9600] loss: 0.299
without reg [5,  9600] loss: 0.031
with reg [5, 12800] loss: 0.302
without reg [5, 12800] loss: 0.034
with reg [5, 16000] loss: 0.302
without reg [5, 16000] loss: 0.036
with reg [5, 19200] loss: 0.299
without reg [5, 19200] loss: 0.033
with reg [5, 22400] loss: 0.301
without reg [5, 22400] loss: 0.036

 25%|██▌       | 5/20 [04:26<13:22, 53.53s/it]

with reg [6,  3200] loss: 0.294
without reg [6,  3200] loss: 0.031
with reg [6,  6400] loss: 0.297
without reg [6,  6400] loss: 0.035
with reg [6,  9600] loss: 0.291
without reg [6,  9600] loss: 0.030
with reg [6, 12800] loss: 0.285
without reg [6, 12800] loss: 0.025
with reg [6, 16000] loss: 0.295
without reg [6, 16000] loss: 0.036
with reg [6, 19200] loss: 0.299
without reg [6, 19200] loss: 0.040
with reg [6, 22400] loss: 0.285
without reg [6, 22400] loss: 0.027

 30%|███       | 6/20 [05:19<12:26, 53.35s/it]
```

```
with reg [7,  3200] loss: 0.277
without reg [7,  3200] loss: 0.021
with reg [7,  6400] loss: 0.283
without reg [7,  6400] loss: 0.028
with reg [7,  9600] loss: 0.282
without reg [7,  9600] loss: 0.027
with reg [7, 12800] loss: 0.280
without reg [7, 12800] loss: 0.027
with reg [7, 16000] loss: 0.284
without reg [7, 16000] loss: 0.031
with reg [7, 19200] loss: 0.281
without reg [7, 19200] loss: 0.030
with reg [7, 22400] loss: 0.281
without reg [7, 22400] loss: 0.030

 35%|███▌      | 7/20 [06:12<11:30, 53.11s/it]

with reg [8,  3200] loss: 0.272
without reg [8,  3200] loss: 0.023
with reg [8,  6400] loss: 0.278
without reg [8,  6400] loss: 0.030
with reg [8,  9600] loss: 0.267
without reg [8,  9600] loss: 0.020
with reg [8, 12800] loss: 0.268
without reg [8, 12800] loss: 0.021
with reg [8, 16000] loss: 0.267
without reg [8, 16000] loss: 0.021
with reg [8, 19200] loss: 0.277
without reg [8, 19200] loss: 0.032
with reg [8, 22400] loss: 0.268
without reg [8, 22400] loss: 0.024

 40%|████      | 8/20 [07:05<10:36, 53.01s/it]

with reg [9,  3200] loss: 0.265
without reg [9,  3200] loss: 0.023
with reg [9,  6400] loss: 0.264
without reg [9,  6400] loss: 0.022
with reg [9,  9600] loss: 0.262
without reg [9,  9600] loss: 0.021
with reg [9, 12800] loss: 0.259
without reg [9, 12800] loss: 0.019
with reg [9, 16000] loss: 0.255
without reg [9, 16000] loss: 0.015
with reg [9, 19200] loss: 0.265
without reg [9, 19200] loss: 0.026
with reg [9, 22400] loss: 0.266
without reg [9, 22400] loss: 0.028

 45%|████▌     | 9/20 [07:57<09:42, 52.92s/it]
```

```
with reg [10,  3200] loss: 0.253
without reg [10,  3200] loss: 0.017
with reg [10,  6400] loss: 0.256
without reg [10,  6400] loss: 0.021
with reg [10,  9600] loss: 0.252
without reg [10,  9600] loss: 0.017
with reg [10, 12800] loss: 0.255
without reg [10, 12800] loss: 0.021
with reg [10, 16000] loss: 0.252
without reg [10, 16000] loss: 0.020
with reg [10, 19200] loss: 0.256
without reg [10, 19200] loss: 0.024
with reg [10, 22400] loss: 0.257
without reg [10, 22400] loss: 0.025
```

```
 50%|█████     | 10/20 [08:51<08:50, 53.04s/it]
```

```
with reg [11,  3200] loss: 0.246
without reg [11,  3200] loss: 0.016
with reg [11,  6400] loss: 0.243
without reg [11,  6400] loss: 0.014
with reg [11,  9600] loss: 0.248
without reg [11,  9600] loss: 0.020
with reg [11, 12800] loss: 0.248
without reg [11, 12800] loss: 0.020
with reg [11, 16000] loss: 0.248
without reg [11, 16000] loss: 0.022
with reg [11, 19200] loss: 0.250
without reg [11, 19200] loss: 0.024
with reg [11, 22400] loss: 0.242
without reg [11, 22400] loss: 0.017
```

```
 55%|█████▌    | 11/20 [09:45<07:59, 53.27s/it]
```

```
with reg [12,  3200] loss: 0.238
without reg [12,  3200] loss: 0.014
with reg [12,  6400] loss: 0.239
without reg [12,  6400] loss: 0.016
with reg [12,  9600] loss: 0.242
without reg [12,  9600] loss: 0.019
with reg [12, 12800] loss: 0.238
without reg [12, 12800] loss: 0.016
with reg [12, 16000] loss: 0.239
without reg [12, 16000] loss: 0.018
with reg [12, 19200] loss: 0.241
without reg [12, 19200] loss: 0.021
with reg [12, 22400] loss: 0.239
without reg [12, 22400] loss: 0.020
```

```
 60%|██████    | 12/20 [10:37<07:04, 53.07s/it]
```

```
with reg [13,  3200] loss: 0.235
without reg [13,  3200] loss: 0.017
with reg [13,  6400] loss: 0.232
without reg [13,  6400] loss: 0.014
with reg [13,  9600] loss: 0.231
without reg [13,  9600] loss: 0.014
with reg [13, 12800] loss: 0.237
without reg [13, 12800] loss: 0.021
with reg [13, 16000] loss: 0.230
without reg [13, 16000] loss: 0.015
with reg [13, 19200] loss: 0.229
without reg [13, 19200] loss: 0.015
with reg [13, 22400] loss: 0.230
without reg [13, 22400] loss: 0.016
```

```
 65%|██████▌   | 13/20 [11:30<06:11, 53.08s/it]
```

```
with reg [14,  3200] loss: 0.227
without reg [14,  3200] loss: 0.014
with reg [14,  6400] loss: 0.224
without reg [14,  6400] loss: 0.012
with reg [14,  9600] loss: 0.228
without reg [14,  9600] loss: 0.017
with reg [14, 12800] loss: 0.229
without reg [14, 12800] loss: 0.019
with reg [14, 16000] loss: 0.230
without reg [14, 16000] loss: 0.020
with reg [14, 19200] loss: 0.227
without reg [14, 19200] loss: 0.018
with reg [14, 22400] loss: 0.226
without reg [14, 22400] loss: 0.017
```

```
 70%|███████   | 14/20 [12:23<05:17, 52.89s/it]
```

```
with reg [15,  3200] loss: 0.223
without reg [15,  3200] loss: 0.015
with reg [15,  6400] loss: 0.220
without reg [15,  6400] loss: 0.013
with reg [15,  9600] loss: 0.223
without reg [15,  9600] loss: 0.016
with reg [15, 12800] loss: 0.221
without reg [15, 12800] loss: 0.015
with reg [15, 16000] loss: 0.222
without reg [15, 16000] loss: 0.017
with reg [15, 19200] loss: 0.219
without reg [15, 19200] loss: 0.014
with reg [15, 22400] loss: 0.217
without reg [15, 22400] loss: 0.013
```

```
 75%|███████▌  | 15/20 [13:16<04:24, 52.96s/it]
```

```
with reg [16,  3200] loss: 0.216
without reg [16,  3200] loss: 0.013
with reg [16,  6400] loss: 0.214
without reg [16,  6400] loss: 0.012
with reg [16,  9600] loss: 0.215
without reg [16,  9600] loss: 0.013
with reg [16, 12800] loss: 0.219
without reg [16, 12800] loss: 0.018
with reg [16, 16000] loss: 0.214
without reg [16, 16000] loss: 0.013
with reg [16, 19200] loss: 0.217
without reg [16, 19200] loss: 0.017
with reg [16, 22400] loss: 0.216
without reg [16, 22400] loss: 0.017
```

```
 80%|████████  | 16/20 [14:09<03:31, 52.91s/it]
```

```
with reg [17,  3200] loss: 0.212
without reg [17,  3200] loss: 0.014
with reg [17,  6400] loss: 0.209
without reg [17,  6400] loss: 0.012
with reg [17,  9600] loss: 0.215
without reg [17,  9600] loss: 0.018
with reg [17, 12800] loss: 0.210
without reg [17, 12800] loss: 0.014
with reg [17, 16000] loss: 0.209
without reg [17, 16000] loss: 0.013
with reg [17, 19200] loss: 0.211
without reg [17, 19200] loss: 0.016
with reg [17, 22400] loss: 0.211
without reg [17, 22400] loss: 0.016
```

```
 85%|████████▌ | 17/20 [15:02<02:38, 52.97s/it]
```

```
with reg [18,  3200] loss: 0.206
without reg [18,  3200] loss: 0.012
with reg [18,  6400] loss: 0.208
without reg [18,  6400] loss: 0.014
with reg [18,  9600] loss: 0.206
without reg [18,  9600] loss: 0.013
with reg [18, 12800] loss: 0.207
without reg [18, 12800] loss: 0.015
with reg [18, 16000] loss: 0.205
without reg [18, 16000] loss: 0.014
with reg [18, 19200] loss: 0.213
without reg [18, 19200] loss: 0.022
with reg [18, 22400] loss: 0.205
without reg [18, 22400] loss: 0.015
```

```
 90%|█████████ | 18/20 [15:55<01:46, 53.01s/it]
```

```
with reg [19,  3200] loss: 0.202
without reg [19,  3200] loss: 0.012
with reg [19,  6400] loss: 0.200
without reg [19,  6400] loss: 0.011
with reg [19,  9600] loss: 0.199
without reg [19,  9600] loss: 0.011
with reg [19, 12800] loss: 0.201
without reg [19, 12800] loss: 0.013
with reg [19, 16000] loss: 0.200
without reg [19, 16000] loss: 0.012
with reg [19, 19200] loss: 0.200
without reg [19, 19200] loss: 0.013
with reg [19, 22400] loss: 0.202
without reg [19, 22400] loss: 0.016
```

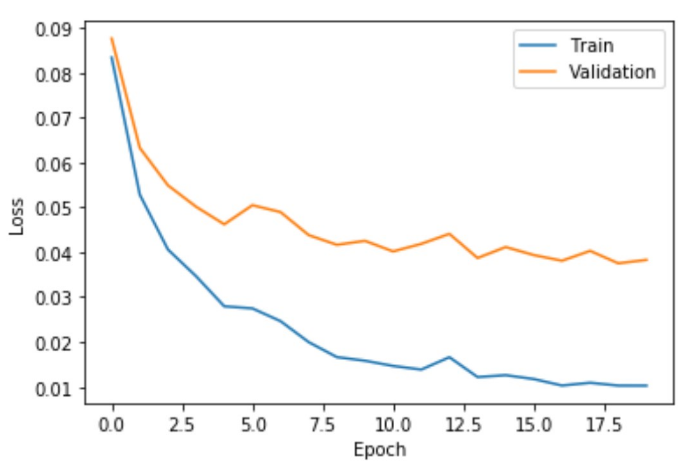
```
 95%|█████████▌| 19/20 [16:48<00:52, 52.98s/it]
```

```
with reg [20,  3200] loss: 0.197
without reg [20,  3200] loss: 0.011
with reg [20,  6400] loss: 0.200
without reg [20,  6400] loss: 0.015
with reg [20,  9600] loss: 0.195
without reg [20,  9600] loss: 0.011
with reg [20, 12800] loss: 0.202
without reg [20, 12800] loss: 0.018
with reg [20, 16000] loss: 0.196
without reg [20, 16000] loss: 0.013
with reg [20, 19200] loss: 0.198
without reg [20, 19200] loss: 0.015
with reg [20, 22400] loss: 0.197
without reg [20, 22400] loss: 0.014
```

```
100%|██████████| 20/20 [17:40<00:00, 52.82s/it]
```

```
Finished Training
Training data accuracy:   99.88788788788789
Validation data accuracy: 98.8457711442786
Testing data accuracy:    98.66
Model Saved
Final validation loss: 0.03827444463968277
```

Out[0]: <matplotlib.legend.Legend at 0x7fa60b8becf8>



For this task, the final 10 models are displayed in the table below. During exploration of the models, we experimented with both the architecture and the hyperparameters of the model.

- **Architecture:** we experimented with the depth and number of channels of the convolutional layers as well as the depth and the number of neurons in each classification layer. We tried 1 to 4 convolutional layers as well as 1 to 4 classification layers and observed that the best results were obtained when there were 3 convolutional layers and 1 classification layers. We observed that our model started to overfit when using many classification layers. We then experimented with different pooling operations (max, mean) and different filter sizes. From our experimentation we find that a max pool layer of size (2,2) and stride 1 always gives better results. We also notice changing the stride or kernel size doesn't lead to any significant improvement in the validation loss, hence in the next phase of experimentation we keep the kernel size fixed at (3,3) in all convolution layers and use a stride of 1. Therefore, in our final table we focused on iteratively changing the number of channels and the number of neurons for the convolution, and the total number of classification layers. We observed that using more than 128 neurons in the classification layers and more than 20 channels in the convolutional layers did not lead to any improvement on the validation loss and sometimes in fact resulted in an increased validation loss, implying that the model was overfitting to the training data.
- **Hyperparameters (related to: activation function, optimiser, regularisation, batchsize):**
  - *Activation function:* First we considered the activation function both in the convolutional and classification layers, experimenting with Relu, logsitic and tanh functions, while iteratively changing the architecture and hyperparameters to best observe the effect of each of the activations. We observed that the Relu functions gave significantly better results and hence decided to use it for the rest of the exploration.
  - *Optimiser & Learning rate:* Next, we considered the 'SGD' and 'ADAM' optimisers, while observing that ADAM with learning rate equal to 0.0001 and SGD with learning rate equal to 0.001 gave the best results on the validation test for most of the times. Although the models using the ADAM optimiser gave the lowest validation loss we observed that the graph of the training and validation losses was smoother and more stable in the case of the SGD optimiser while also achieving very low validation loss.
  - *Regularisation (and regularisation parameter λ):* Furthermore, we explored two types of regularisation, L1 and L2. Models using L2 regularisation tended to perform better overall, but both regularistions outperformed the models without regularisation. We considered the regularisation parameter λ by picking values form 0.0001 to 0.1. We concluded that a value of $\lambda = 0.01$ in L2 regularisation gave the best results.
  - *Batchsize:* Lastly, we considered different batchsizes, by adjusting the batch size throughout to find out a resonable number of batches to deliver, which balances good results while keeping the training time reasonably low. We concluded that using a batchsize of 32 made training efficient, while also leading to good results on the validation set.

In [0]:
```
# Append results to the data frame above

df = train_and_append(df, [28,28], [4,4], [[2,2],[2,2]], [3,3], [32], 5, 0.001, 1,
0.001, 20, 'SGD')  # model 2
df = train_and_append(df, [28,28], [10,10,10], [[2,2],[2,2],[2,2]], [3,3,3], [64],
5, 0.001, 2, 0.001, 20, 'SGD') # model 3
df = train_and_append(df, [28,28], [10,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.001, 1, 0.001, 20, 'SGD') # model 4
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [64,3
2], 5, 0.001, 2, 0.01, 20, 'SGD') # model 5
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:16<05:17, 16.70s/it]
 10%|█         | 2/20 [00:32<04:57, 16.54s/it]
 15%|█▌        | 3/20 [00:48<04:39, 16.41s/it]
 20%|██        | 4/20 [01:05<04:21, 16.34s/it]
 25%|██▌       | 5/20 [01:21<04:04, 16.28s/it]
 30%|███       | 6/20 [01:37<03:49, 16.38s/it]
 35%|███▌      | 7/20 [01:56<03:41, 17.01s/it]
 40%|████      | 8/20 [02:14<03:27, 17.31s/it]
 45%|████▌     | 9/20 [02:32<03:13, 17.63s/it]
 50%|█████     | 10/20 [02:49<02:54, 17.47s/it]
 55%|█████▌    | 11/20 [03:06<02:36, 17.34s/it]
 60%|██████    | 12/20 [03:24<02:19, 17.44s/it]
 65%|██████▌   | 13/20 [03:41<02:01, 17.37s/it]
 70%|███████   | 14/20 [03:58<01:42, 17.03s/it]
 75%|███████▌  | 15/20 [04:14<01:24, 16.88s/it]
 80%|████████  | 16/20 [04:31<01:07, 16.86s/it]
 85%|████████▌ | 17/20 [04:49<00:51, 17.25s/it]
 90%|█████████ | 18/20 [05:06<00:34, 17.03s/it]
 95%|█████████▌| 19/20 [05:22<00:16, 16.82s/it]
100%|██████████| 20/20 [05:38<00:00, 16.64s/it]

  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:35<11:11, 35.33s/it]
 10%|█         | 2/20 [01:08<10:24, 34.69s/it]
 15%|█▌        | 3/20 [01:39<09:31, 33.63s/it]
 20%|██        | 4/20 [02:10<08:45, 32.82s/it]
 25%|██▌       | 5/20 [02:40<08:00, 32.05s/it]
 30%|███       | 6/20 [03:11<07:23, 31.71s/it]
 35%|███▌      | 7/20 [03:42<06:48, 31.43s/it]
 40%|████      | 8/20 [04:13<06:14, 31.20s/it]
 45%|████▌     | 9/20 [04:43<05:40, 30.92s/it]
 50%|█████     | 10/20 [05:14<05:08, 30.89s/it]
 55%|█████▌    | 11/20 [05:45<04:38, 30.90s/it]
 60%|██████    | 12/20 [06:16<04:07, 30.93s/it]
 65%|██████▌   | 13/20 [06:47<03:36, 30.92s/it]
 70%|███████   | 14/20 [07:17<03:04, 30.68s/it]
 75%|███████▌  | 15/20 [07:47<02:32, 30.57s/it]
 80%|████████  | 16/20 [08:17<02:01, 30.46s/it]
 85%|████████▌ | 17/20 [08:48<01:31, 30.49s/it]
 90%|█████████ | 18/20 [09:18<01:01, 30.52s/it]
 95%|█████████▌| 19/20 [09:49<00:30, 30.50s/it]
100%|██████████| 20/20 [10:19<00:00, 30.48s/it]

  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:33<10:29, 33.14s/it]
 10%|█         | 2/20 [01:05<09:53, 32.95s/it]
 15%|█▌        | 3/20 [01:37<09:16, 32.74s/it]
 20%|██        | 4/20 [02:11<08:45, 32.87s/it]
 25%|██▌       | 5/20 [02:44<08:16, 33.12s/it]
 30%|███       | 6/20 [03:17<07:42, 33.04s/it]
 35%|███▌      | 7/20 [03:50<07:08, 32.99s/it]
 40%|████      | 8/20 [04:22<06:33, 32.77s/it]
 45%|████▌     | 9/20 [04:55<06:00, 32.80s/it]
 50%|█████     | 10/20 [05:28<05:27, 32.80s/it]
 55%|█████▌    | 11/20 [06:00<04:52, 32.52s/it]
 60%|██████    | 12/20 [06:32<04:20, 32.57s/it]
 65%|██████▌   | 13/20 [07:05<03:47, 32.55s/it]
 70%|███████   | 14/20 [07:38<03:15, 32.62s/it]
 75%|███████▌  | 15/20 [08:11<02:43, 32.72s/it]
 80%|████████  | 16/20 [08:43<02:10, 32.65s/it]
 85%|████████▌ | 17/20 [09:16<01:37, 32.65s/it]
 90%|█████████ | 18/20 [09:49<01:05, 32.65s/it]
 95%|█████████▌| 19/20 [10:21<00:32, 32.59s/it]
```

In [0]:

```
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.001, 2, 0.01, 20, 'SGD') # model 6
df = train_and_append(df, [28,28], [10,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [64,6
4], 5, 0.001, 2, 0.001, 20, 'ADAM') # model 7
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128,1
28], 5, 0.0001, 2, 0.01, 20, 'ADAM') # model 8
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.0001, 2, 0.01, 20, 'ADAM') #model 9
df = train_and_append(df, [28,28], [20,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.0001, 2, 0.01, 20, 'ADAM') # model 10
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:38<12:19, 38.90s/it]
 10%|█         | 2/20 [01:16<11:33, 38.52s/it]
 15%|█▌        | 3/20 [01:55<10:54, 38.52s/it]
 20%|██        | 4/20 [02:32<10:11, 38.24s/it]
 25%|██▌       | 5/20 [03:10<09:32, 38.16s/it]
 30%|███       | 6/20 [03:48<08:52, 38.01s/it]
 35%|███▌      | 7/20 [04:26<08:13, 37.96s/it]
 40%|████      | 8/20 [05:03<07:32, 37.73s/it]
 45%|████▌     | 9/20 [05:40<06:54, 37.67s/it]
 50%|█████     | 10/20 [06:18<06:16, 37.69s/it]
 55%|█████▌    | 11/20 [06:56<05:38, 37.63s/it]
 60%|██████    | 12/20 [07:33<05:01, 37.67s/it]
 65%|██████▌   | 13/20 [08:11<04:22, 37.57s/it]
 70%|███████   | 14/20 [08:48<03:45, 37.59s/it]
 75%|███████▌  | 15/20 [09:26<03:08, 37.76s/it]
 80%|████████  | 16/20 [10:04<02:31, 37.78s/it]
 85%|████████▌ | 17/20 [10:43<01:53, 37.94s/it]
 90%|█████████ | 18/20 [11:20<01:15, 37.73s/it]
 95%|█████████▌| 19/20 [11:58<00:37, 37.77s/it]
100%|██████████| 20/20 [12:35<00:00, 37.71s/it]

  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:37<11:57, 37.78s/it]
 10%|█         | 2/20 [01:15<11:19, 37.75s/it]
 15%|█▌        | 3/20 [01:53<10:42, 37.77s/it]
 20%|██        | 4/20 [02:30<10:03, 37.70s/it]
 25%|██▌       | 5/20 [03:08<09:27, 37.84s/it]
 30%|███       | 6/20 [03:46<08:48, 37.78s/it]
 35%|███▌      | 7/20 [04:23<08:08, 37.61s/it]
 40%|████      | 8/20 [05:01<07:31, 37.66s/it]
 45%|████▌     | 9/20 [05:39<06:55, 37.74s/it]
 50%|█████     | 10/20 [06:17<06:16, 37.69s/it]
 55%|█████▌    | 11/20 [06:54<05:38, 37.59s/it]
 60%|██████    | 12/20 [07:32<05:00, 37.59s/it]
 65%|██████▌   | 13/20 [08:09<04:23, 37.68s/it]
 70%|███████   | 14/20 [08:47<03:46, 37.73s/it]
 75%|███████▌  | 15/20 [09:27<03:10, 38.20s/it]
 80%|████████  | 16/20 [10:05<02:33, 38.29s/it]
 85%|████████▌ | 17/20 [10:44<01:55, 38.46s/it]
 90%|█████████ | 18/20 [11:23<01:17, 38.78s/it]
 95%|█████████▌| 19/20 [12:02<00:38, 38.83s/it]
100%|██████████| 20/20 [12:41<00:00, 38.87s/it]

  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:39<12:38, 39.91s/it]
 10%|█         | 2/20 [01:19<11:58, 39.89s/it]
 15%|█▌        | 3/20 [01:59<11:19, 39.97s/it]
 20%|██        | 4/20 [02:40<10:40, 40.05s/it]
 25%|██▌       | 5/20 [03:20<10:02, 40.16s/it]
 30%|███       | 6/20 [04:01<09:23, 40.25s/it]
 35%|███▌      | 7/20 [04:41<08:43, 40.27s/it]
 40%|████      | 8/20 [05:21<08:03, 40.26s/it]
 45%|████▌     | 9/20 [06:01<07:23, 40.28s/it]
 50%|█████     | 10/20 [06:42<06:42, 40.24s/it]
 55%|█████▌    | 11/20 [07:23<06:05, 40.57s/it]
 60%|██████    | 12/20 [08:03<05:23, 40.44s/it]
 65%|██████▌   | 13/20 [08:43<04:42, 40.34s/it]
 70%|███████   | 14/20 [09:24<04:02, 40.39s/it]
 75%|███████▌  | 15/20 [10:04<03:21, 40.26s/it]
 80%|████████  | 16/20 [10:44<02:41, 40.37s/it]
 85%|████████▌ | 17/20 [11:25<02:01, 40.39s/it]
 90%|█████████ | 18/20 [12:05<01:20, 40.45s/it]
 95%|█████████▌| 19/20 [12:46<00:40, 40.43s/it]
```

In [0]:
```
display(df)
```

Out[0]:

	Convolution layers	Classification hidden units	Optimiser	Learning rate	Regularisation/ parameter (lambda)	Train_Loss	Valid_Loss	Test_Loss	Train_
1	[4, 4]	[128]	SGD	0.0010	L 1, lambda = 0	0.066362	0.079999	0.073290	9
2	[4, 4]	[32]	SGD	0.0010	L 1, lambda = 0.001	0.072042	0.074005	0.077724	9
3	[10, 10, 10]	[64]	SGD	0.0010	L 2, lambda = 0.001	0.048030	0.063892	0.070390	9
4	[10, 15, 15]	[128]	SGD	0.0010	L 1, lambda = 0.001	0.047329	0.062488	0.062052	9
6	[15, 15, 15]	[64, 32]	SGD	0.0010	L 2, lambda = 0.01	0.033215	0.055223	0.052725	9
7	[15, 15, 15]	[128]	SGD	0.0010	L 2, lambda = 0.01	0.036966	0.052174	0.053685	9
5	[10, 20, 20]	[64, 64]	ADAM	0.0010	L 2, lambda = 0.001	0.006986	0.049017	0.052544	9
9	[15, 15, 15]	[128, 128]	ADAM	0.0001	L 2, lambda = 0.01	0.013971	0.046347	0.044284	9
8	[15, 15, 15]	[128]	ADAM	0.0001	L 2, lambda = 0.01	0.016603	0.044719	0.042251	9
10	[20, 20, 20]	[128]	ADAM	0.0001	L 2, lambda = 0.01	0.011473	0.037712	0.036817	9

	Convolution layers	Classification hidden units	Optimiser	Learning rate	Regularisation/ parameter (lambda)	Train_Loss	Valid_Loss	Test_Loss	Train_accuracy (%)	Valid_accuracy (%)	Test_accuracy (%)
1	[4, 4]	[128]	SGD	0.0010	L 1, lambda = 0	0.066362	0.079999	0.073290	98.182182	97.393035	97.68
2	[4, 4]	[32]	SGD	0.0010	L 1, lambda = 0.001	0.072042	0.074005	0.077724	98.066066	97.990050	97.74
3	[10, 10, 10]	[64]	SGD	0.0010	L 2, lambda = 0.001	0.048030	0.063892	0.070390	98.666667	97.930348	97.86
4	[10, 15, 15]	[128]	SGD	0.0010	L 1, lambda = 0.001	0.047329	0.062488	0.062052	98.874875	98.248756	98.10
6	[15, 15, 15]	[64, 32]	SGD	0.0010	L 2, lambda = 0.01	0.033215	0.055223	0.052725	99.135135	98.388060	98.46
7	[15, 15, 15]	[128]	SGD	0.0010	L 2, lambda = 0.01	0.036966	0.052174	0.053685	99.107107	98.507463	98.48
5	[10, 20, 20]	[64, 64]	ADAM	0.0010	L 2, lambda = 0.001	0.006986	0.049017	0.052544	99.827828	98.766169	98.68
9	[15, 15, 15]	[128, 128]	ADAM	0.0001	L 2, lambda = 0.01	0.013971	0.046347	0.044284	99.783784	98.646766	98.68
8	[15, 15, 15]	[128]	ADAM	0.0001	L 2, lambda = 0.01	0.016603	0.044719	0.042251	99.759760	98.606965	98.72
10	[20, 20, 20]	[128]	ADAM	0.0001	L 2, lambda = 0.01	0.011473	0.037712	0.036817	99.831832	98.845771	98.88

3. Was the lowest test loss obtained for model with the lowest validation loss? If not, why do you think this was the case?

The lowest test loss was obtained for model with the lowest validation loss. However, we did find that using more complex models, for example with number of hidden layers greater than 128 resulted in a decrease in training loss but an increase in validation and test losses, suggesting that the model was not generalising well. Hence, we find that more complex do not necessarily outperform simpler ones.

In [0]:

Q3

Question 3

In [0]:
```
import os
import gzip
import numpy as np
os.system("pip install python-mnist")
from mnist import MNIST
import matplotlib.pyplot as plt
%matplotlib inline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.autograd import Variable
from sklearn.metrics import confusion_matrix
import pandas as pd
from tqdm import tqdm
```

In [0]:
```
# Download the dataset.
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-images-idx3-ubyte.gz -o /tmp/train-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-labels-idx1-ubyte.gz -o /tmp/train-labels-idx1-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-images-idx3-ubyte.gz -o /tmp/t10k-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-labels-idx1-ubyte.gz -o /tmp/t10k-labels-idx1-ubyte.gz')
pass
```

In [0]:
```
# load the dataset with shape (*, 784)
mnistdata = MNIST("/tmp")
mnistdata.gz = True
trainxs_raw, trainys_raw = mnistdata.load_training()
testxs_raw, testys_raw = mnistdata.load_testing()

# reshape data into square image (*, 1, 28, 28)
trainxs_square  = np.array(trainxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
trainys_square  = np.array(trainys_raw, dtype=np.long)
testxs          = np.array(testxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
testys          = np.array(testys_raw, dtype=np.long)

# split training dataset
trainxs = trainxs_square[:50000]
trainys = trainys_square[:50000]
validxs = trainxs_square[50000:]
validys = trainys_square[50000:]
```

In [0]:
```
# process into FashionMNist 1 and Fashon MNist 2

def filter_for_labels(trainxs, trainys, validxs, validys, testxs, testys, labels):
    # FashinoMNist 1
    train_ix = np.isin(trainys, labels)
    trainxs_ = trainxs[train_ix]
    trainys_ = trainys[train_ix]

    val_ix = np.isin(validys, labels)
    validxs_ = validxs[val_ix]
    validys_ = validys[val_ix]

    test_ix = np.isin(testys, labels)
    testxs_ = testxs[test_ix]
    testys_ = testys[test_ix]

    return trainxs_, trainys_, validxs_, validys_, testxs_, testys_

labels_1 = [0, 1, 4, 5, 8]
labels_2 = [x for x in list(range(10)) if x not in labels_1]

# FashinoMNist1
trainxs_1, trainys_1, validxs_1, validys_1, testxs_1, testys_1 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_1
)

def change_labels(ys, labels):
  new_ys = np.zeros(len(ys))
  position_dict = {labels[i]: i for i in range(len(labels))}
  for label in position_dict.keys():
    new_ys[ys == label] = position_dict[label]
  return new_ys

trainys_1 = change_labels(trainys_1, labels_1)
validys_1 = change_labels(validys_1, labels_1)
testys_1 = change_labels(testys_1, labels_1)

trainys_1 = np.array(trainys_1, dtype=np.long)
validys_1 = np.array(validys_1, dtype=np.long)
testys_1 = np.array(testys_1, dtype=np.long)


#FashionMNist2
trainxs_2, trainys_2, validxs_2, validys_2, testxs_2, testys_2 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_2
)
```

In [0]:
```
class DatasetConstructor(Dataset):

    def __init__(self, X, Y):
        self.X = torch.from_numpy(X)
        self.Y = torch.from_numpy(Y)
        self.len = X.shape[0]

    def __getitem__(self, index):
        return self.X[index], self.Y[index]

    def __len__(self):
        return self.len

BATCHSIZE = 32

train_data = DatasetConstructor(trainxs_1, trainys_1)
valid_data = DatasetConstructor(validxs_1, validys_1)
test_data  = DatasetConstructor(testxs_1, testys_1)

train_loader  = DataLoader(dataset=train_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
valid_loader  = DataLoader(dataset=valid_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
test_loader   = DataLoader(dataset=test_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)

# tensor dataset for loss and accuracy computation
trainxs_tensor = torch.from_numpy(trainxs_1)
trainys_tensor = torch.from_numpy(trainys_1)
validxs_tensor = torch.from_numpy(validxs_1)
validys_tensor = torch.from_numpy(validys_1)
testxs_tensor  = torch.from_numpy(testxs_1)
testys_tensor  = torch.from_numpy(testys_1)
```

In [0]:
```
# dataset examples
fig=plt.figure(figsize=(20, 5))
columns = 20
rows = 5
for i in range(columns*rows):
    fig.add_subplot(rows, columns, i+1)
    plt.axis('off')
    plt.imshow(trainxs_1[i, 0], cmap='binary')
plt.show()
```

1. Implement a multi-class, convolutional neural network with cross-entropy loss for the Fashion-MNIST-1 data.

In [0]:
```
# Class to define model of arbitrarily sized architecture

class CNN_class(nn.Module):

    def __init__(self, input_size, channels_conv, pool_size, kernel_size, hids, out
put_size):
        super(CNN_class, self).__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.channels_conv = channels_conv
        self.kernel_size = kernel_size
        self.pool_size = pool_size
        self.hids = hids
        self.conv_layers = []
        self.classification_layers = []

        assert len(channels_conv) > 0
        assert len(hids) > 0
        assert len(channels_conv) == len(kernel_size) == len(pool_size)
        assert all(i > 0 for i in hids)
        assert all(i > 0 for i in channels_conv)
        assert all(i > 0 for i in kernel_size)

        self.conv_layers.append(nn.Conv2d(1, self.channels_conv[0], self.kernel_siz
e[0], padding=1, padding_mode="same"))
        self.conv_layers.append(nn.BatchNorm2d(self.channels_conv[0]))
        self.conv_layers.append(nn.ReLU(inplace=True))
        self.conv_layers.append(nn.MaxPool2d(self.pool_size[0],self.pool_size[0]))

        for i in range(1,len(self.channels_conv)):
          self.conv_layers.append(nn.Conv2d(self.channels_conv[i-1], self.channels_
conv[i], self.kernel_size[i], padding=1, padding_mode="same"))
          self.conv_layers.append(nn.BatchNorm2d(self.channels_conv[i]))
          self.conv_layers.append(nn.ReLU(inplace=True))
          self.conv_layers.append(nn.MaxPool2d(self.pool_size[i],self.pool_size
[i]))

        self.conv = nn.Sequential(*self.conv_layers)

        p = np.array(self.input_size) // self.pool_size[0][0]
        for i in range(1,len(self.channels_conv)):
          p = p // self.pool_size[i][0]

        self.conv_out_size = int(p[0]*p[1]*self.channels_conv[-1])

        self.classification_layers.append(nn.Linear(self.conv_out_size, self.hids
[0]))
        self.classification_layers.append(nn.ReLU(inplace=True))

        for i in range(1,len(hids)):
          self.classification_layers.append(nn.Linear(self.hids[i-1], self.hids
[i]))
          self.classification_layers.append(nn.ReLU(inplace=True))

        self.classification_layers.append(nn.Linear(self.hids[-1], self.output_siz
e))

        self.classif = nn.Sequential(*self.classification_layers)

    def forward(self, x):

      x = self.conv(x)
      x = x.view(x.size(0), -1)      # reshape
      x = self.classif(x)
```

In [0]:
```
def accuracy(model, X, Y):
    outputs = model(X)
    _, prediction = torch.max(outputs, 1)
    if torch.cuda.is_available():
        prediction = prediction.cpu()
    return sum((Y - prediction.numpy())==0) / len(Y) * 100
```

2. Train your final model to convergence on the training set using an optimisation algorithm of your choice.

In [0]:
```
# Training function for input training parameters

def train_CNN(model, lr, reg, lambd, num_of_epochs, opt=''):

  # Generic function to train model with input training parameters:
  # lr = learning rate
  # reg = 1 or 2 for L1 or L2 regularisation
  # lambd = parameter lambda of regularisation - set to lambda = 0 for no regularis
ation
  # num_of_epochs = number of epochs for training loop
  # opt = keyword argument set to SGD or ADAM

  if torch.cuda.is_available():
    model.cuda()

  loss = nn.CrossEntropyLoss()
  if opt == 'SGD':
    optimizer = optim.SGD(model.parameters(), lr=lr)
  else:
    optimizer = optim.Adam(model.parameters(), lr=lr)

  train_loss = []
  valid_loss = []
  test_loss = []

  for epoch in tqdm(range(num_of_epochs)):

    running_loss = 0.0
    running_loss_without_reg = 0.0
    for i, data in enumerate(train_loader, 0):
      inputs, labels = data

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = model(inputs)
      loss_val_pre = loss(outputs, labels)
      regularisation = 0
      for param in model.parameters():
          regularisation += torch.norm(param, reg)
      loss_val = loss_val_pre + lambd*regularisation
      loss_val.backward()
      optimizer.step()

      # print statistics
      running_loss += loss_val.item()
      running_loss_without_reg += loss_val_pre.item()
      if i % 200 == 199:    # print every 200 mini-batches
          print('with reg [%d, %5d] loss: %.3f' %
                (epoch + 1, (i + 1)*BATCHSIZE, running_loss / 200))
          running_loss = 0.0
          print('without reg [%d, %5d] loss: %.3f' %
                (epoch + 1, (i + 1)*BATCHSIZE, running_loss_without_reg / 200))
          running_loss_without_reg = 0.0

    if torch.cuda.is_available():
        train_loss.append(loss(model(trainxs_tensor.cuda()), trainys_tensor.cuda
()).item())
        valid_loss.append(loss(model(validxs_tensor.cuda()), validys_tensor.cuda
()).item())
```

In [0]:
```
# Final model from task 2
input_size = [28,28]
channels_conv = [20,20,20]
pool_size = [[2,2], [2,2], [2,2]]
kernel_size = [3,3,3]
hids = [128]
output_size = 5
lr = 0.0001
reg = 2
lmbd = 0.01
num_of_epochs = 20
opt = 'ADAM'

model = CNN_class(input_size, channels_conv, pool_size, kernel_size, hids, output_s
ize)
train_loss, valid_loss, test_loss, final_accuracies = train_CNN(model, lr, reg, lmb
d, num_of_epochs, opt=opt)
```

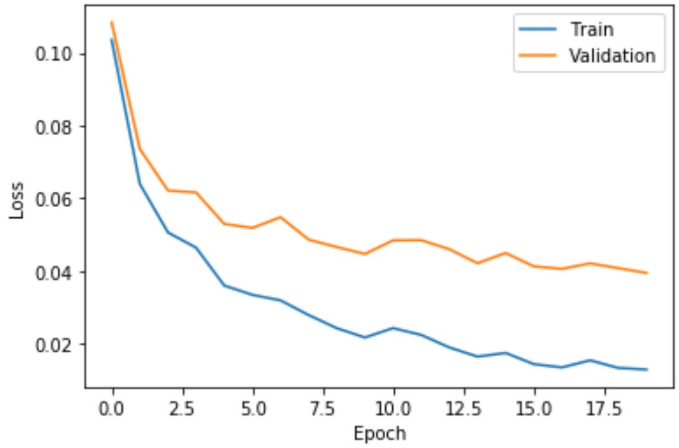
```
  0%|          | 0/20 [00:00<?, ?it/s]

with reg [1,  6400] loss: 1.106
without reg [1,  6400] loss: 0.798
with reg [1, 12800] loss: 0.539
without reg [1, 12800] loss: 0.232
with reg [1, 19200] loss: 0.443
without reg [1, 19200] loss: 0.138

  5%|▌         | 1/20 [00:06<01:59,  6.27s/it]

with reg [2,  6400] loss: 0.399
without reg [2,  6400] loss: 0.098
with reg [2, 12800] loss: 0.383
without reg [2, 12800] loss: 0.084
with reg [2, 19200] loss: 0.377
without reg [2, 19200] loss: 0.081

 10%|█         | 2/20 [00:12<01:52,  6.25s/it]

with reg [3,  6400] loss: 0.359
without reg [3,  6400] loss: 0.067
with reg [3, 12800] loss: 0.354
without reg [3, 12800] loss: 0.064
with reg [3, 19200] loss: 0.346
without reg [3, 19200] loss: 0.057

 15%|█▌        | 3/20 [00:18<01:45,  6.19s/it]

with reg [4,  6400] loss: 0.334
without reg [4,  6400] loss: 0.049
with reg [4, 12800] loss: 0.334
without reg [4, 12800] loss: 0.051
with reg [4, 19200] loss: 0.335
without reg [4, 19200] loss: 0.054

 20%|██        | 4/20 [00:24<01:38,  6.14s/it]

with reg [5,  6400] loss: 0.315
without reg [5,  6400] loss: 0.036
with reg [5, 12800] loss: 0.324
without reg [5, 12800] loss: 0.046
with reg [5, 19200] loss: 0.323
without reg [5, 19200] loss: 0.046

 25%|██▌       | 5/20 [00:30<01:32,  6.17s/it]

with reg [6,  6400] loss: 0.313
without reg [6,  6400] loss: 0.039
with reg [6, 12800] loss: 0.309
without reg [6, 12800] loss: 0.036
with reg [6, 19200] loss: 0.316
without reg [6, 19200] loss: 0.045

 30%|███       | 6/20 [00:36<01:25,  6.13s/it]

with reg [7,  6400] loss: 0.300
without reg [7,  6400] loss: 0.030
with reg [7, 12800] loss: 0.302
without reg [7, 12800] loss: 0.034
with reg [7, 19200] loss: 0.301
without reg [7, 19200] loss: 0.035

 35%|███▌      | 7/20 [00:42<01:19,  6.12s/it]
```

```
with reg [8,  6400] loss: 0.293
without reg [8,  6400] loss: 0.028
with reg [8, 12800] loss: 0.292
without reg [8, 12800] loss: 0.029
with reg [8, 19200] loss: 0.296
without reg [8, 19200] loss: 0.035

 40%|████      | 8/20 [00:48<01:12,  6.03s/it]

with reg [9,  6400] loss: 0.283
without reg [9,  6400] loss: 0.024
with reg [9, 12800] loss: 0.290
without reg [9, 12800] loss: 0.032
with reg [9, 19200] loss: 0.290
without reg [9, 19200] loss: 0.033

 45%|████▌     | 9/20 [00:54<01:05,  5.98s/it]

with reg [10,  6400] loss: 0.280
without reg [10,  6400] loss: 0.026
with reg [10, 12800] loss: 0.278
without reg [10, 12800] loss: 0.025
with reg [10, 19200] loss: 0.281
without reg [10, 19200] loss: 0.029

 50%|█████     | 10/20 [01:00<00:59,  5.92s/it]

with reg [11,  6400] loss: 0.274
without reg [11,  6400] loss: 0.024
with reg [11, 12800] loss: 0.272
without reg [11, 12800] loss: 0.023
with reg [11, 19200] loss: 0.272
without reg [11, 19200] loss: 0.025

 55%|█████▌    | 11/20 [01:06<00:53,  5.93s/it]

with reg [12,  6400] loss: 0.268
without reg [12,  6400] loss: 0.022
with reg [12, 12800] loss: 0.267
without reg [12, 12800] loss: 0.022
with reg [12, 19200] loss: 0.265
without reg [12, 19200] loss: 0.022

 60%|██████    | 12/20 [01:12<00:47,  5.89s/it]

with reg [13,  6400] loss: 0.263
without reg [13,  6400] loss: 0.022
with reg [13, 12800] loss: 0.259
without reg [13, 12800] loss: 0.020
with reg [13, 19200] loss: 0.259
without reg [13, 19200] loss: 0.020

 65%|██████▌   | 13/20 [01:17<00:40,  5.85s/it]

with reg [14,  6400] loss: 0.255
without reg [14,  6400] loss: 0.018
with reg [14, 12800] loss: 0.260
without reg [14, 12800] loss: 0.024
with reg [14, 19200] loss: 0.253
without reg [14, 19200] loss: 0.019

 70%|███████   | 14/20 [01:23<00:35,  5.92s/it]
```

```
with reg [15,  6400] loss: 0.251
without reg [15,  6400] loss: 0.019
with reg [15, 12800] loss: 0.248
without reg [15, 12800] loss: 0.016
with reg [15, 19200] loss: 0.249
without reg [15, 19200] loss: 0.018

 75%|███████▌  | 15/20 [01:29<00:29,  5.92s/it]

with reg [16,  6400] loss: 0.246
without reg [16,  6400] loss: 0.018
with reg [16, 12800] loss: 0.243
without reg [16, 12800] loss: 0.016
with reg [16, 19200] loss: 0.245
without reg [16, 19200] loss: 0.019

 80%|████████  | 16/20 [01:35<00:23,  5.91s/it]

with reg [17,  6400] loss: 0.240
without reg [17,  6400] loss: 0.016
with reg [17, 12800] loss: 0.241
without reg [17, 12800] loss: 0.017
with reg [17, 19200] loss: 0.240
without reg [17, 19200] loss: 0.017

 85%|████████▌ | 17/20 [01:41<00:17,  5.92s/it]

with reg [18,  6400] loss: 0.237
without reg [18,  6400] loss: 0.016
with reg [18, 12800] loss: 0.237
without reg [18, 12800] loss: 0.017
with reg [18, 19200] loss: 0.235
without reg [18, 19200] loss: 0.016

 90%|█████████ | 18/20 [01:47<00:11,  5.90s/it]

with reg [19,  6400] loss: 0.233
without reg [19,  6400] loss: 0.016
with reg [19, 12800] loss: 0.232
without reg [19, 12800] loss: 0.015
with reg [19, 19200] loss: 0.232
without reg [19, 19200] loss: 0.017

 95%|█████████▌| 19/20 [01:53<00:05,  5.92s/it]

with reg [20,  6400] loss: 0.229
without reg [20,  6400] loss: 0.015
with reg [20, 12800] loss: 0.226
without reg [20, 12800] loss: 0.013
with reg [20, 19200] loss: 0.227
without reg [20, 19200] loss: 0.015

100%|██████████| 20/20 [01:59<00:00,  5.89s/it]

Training data accuracy:   99.76376376376376
Validation data accuracy: 98.72636815920399
Testing data accuracy:    98.96000000000001
Model Saved
```

In [0]:
```
fig = plt.figure()
plt.plot(train_loss)
plt.plot(valid_loss)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(['Train', 'Validation'])
```

Out[0]: <matplotlib.legend.Legend at 0x7f01b34a1550>



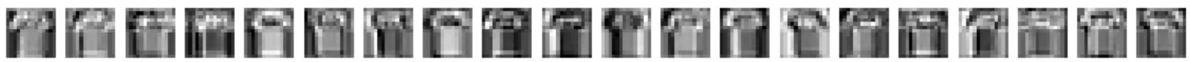
3. Retrieve and visualise the feature maps for each layer of your convolutional neural network.

In [0]:
```
# feature maps visualisation

def visualise_feature_maps(index):
  activation = {}
  def get_features(name):
     def hook(model, input, output):
        activation[name] = output.detach()
     return hook

  data = train_loader.dataset.X[index]
  data.unsqueeze_(0)
  data = data.cuda()

  #for layer in range(len(model.conv_layers)):
    #print(str(layer) + " layer is " + str(model.conv_layers[layer]))

  model.conv_layers[0].register_forward_hook(get_features('conv1'))
  model.conv_layers[2].register_forward_hook(get_features('relu1'))
  model.conv_layers[4].register_forward_hook(get_features('conv2'))
  model.conv_layers[6].register_forward_hook(get_features('relu2'))
  model.conv_layers[8].register_forward_hook(get_features('conv3'))
  model.conv_layers[10].register_forward_hook(get_features('relu3'))

  output = model(data)

  for key in activation.keys():
      print("Feature Maps for", key)
      maps = activation[key].squeeze()
      num = maps.size(0)
      fig=plt.figure(figsize=(num, 1))
      for i in range(num):
          fig.add_subplot(1, num, i+1)
          plt.axis('off')
          plt.imshow(maps[i].cpu(), cmap='binary')
      plt.show()

# one piece of clothing from each class

visualise_feature_maps(0)
visualise_feature_maps(3)
visualise_feature_maps(8)
visualise_feature_maps(10)
visualise_feature_maps(13)
```

Feature Maps for conv1

Feature Maps for relu1

Feature Maps for conv2



Feature Maps for relu2

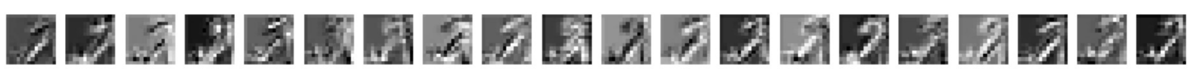
Feature Maps for conv3

Feature Maps for relu3

Feature Maps for conv1

Feature Maps for relu1

Feature Maps for conv2



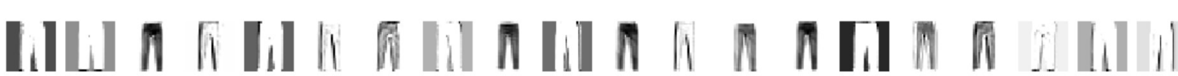
Feature Maps for relu2

Feature Maps for conv3

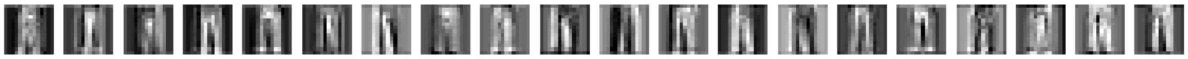
Feature Maps for relu3

Feature Maps for conv1

Feature Maps for relu1

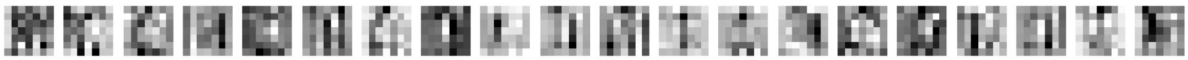


Feature Maps for conv2



Feature Maps for relu2

Feature Maps for conv3

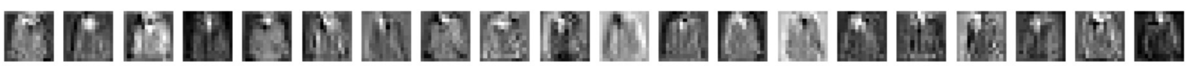


Feature Maps for relu3

Feature Maps for conv1

Feature Maps for relu1

Feature Maps for conv2

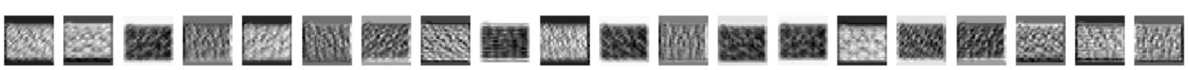


Feature Maps for relu2

Feature Maps for conv3

Feature Maps for relu3

Feature Maps for conv1



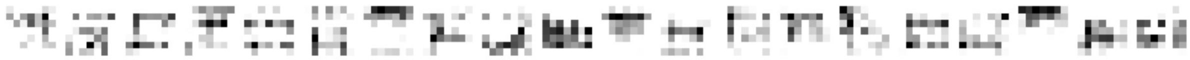
Feature Maps for relu1

Feature Maps for conv2

Feature Maps for relu2

```
Feature Maps for conv3
```

```
Feature Maps for relu3
```



4. Qualitatively analyse the feature maps and hypothesise what they capture and if possible – in particular for the deeper layers – associate them with the output classes.

Above we visualise the feature maps for each of the three convolutional layers, showing the output from both: the convolution operation and the activation.

To achieve each of the feature maps a kernel, or a filter, is applied to every section of the image by performing element-wise matrix multiplication. Next, to reduce dimensionality, we downsample each feature map by performing max-pooling. In each of the convolutional layers feature maps are generated, and the output is passed through an activation function before connecting to the next layer.

To interpret the above-printed maps, it is important to note that the brighter (i.e. whiter) instances of the images correspond to the features that the given filter detected.

We can observe that the images in the first layer are clearer and easily interpretable: the first layers detect edges and general shapes. We also observe that the feature maps in the first layer are much bigger than the ones for the deeper layers - this is because the first layer detects simple features, such as edges, which most of the input images have, hence they retain a lot of the information from the input image. Meanwhile, we notice that the deeper layers become progressively more abstract and sparse. This is because they learn to filter for more complex and specific characteristics, such as a heel or a sleeve - which will only be present in some images.

This also explains why it is easier for humans to interpret the first couple of layers, while for the latter ones it is impossible to interpret them by simple visual analysis.

We will now exemplify the above reasoning on a specific example by considering the feature maps for the first t-shirt. We can see that after the first convolutional layer and ReLu layer (conv1 and relu1) the filter detects the entire piece of clothing. That is, it learns to detect the edges of the clothing item, hence learning its shape. For instance, in the first example above, we see that after the first convolutional layer the tshirt is brighter. After the second convolutional layer, the image becomes blurrier around the print in the centre of the t-shirt and detects the rest. This could be because prints are more likely to be present on tshirts than, for example, on a shoe or on a pair of trousers. Hence, while a tshirt does not necessarily always a print, this feature can nonetheless be indicative of it being a tshirt or not, hence it is detected by the filters. The third layer becomes a lot more sparse and it is very hard to specify what the filter is trying to detect, for the reasons explained in the above paragraph.

```
In [0]:
```

Q4

In [0]:
```
import os
import gzip
import numpy as np
os.system("pip install python-mnist")
from mnist import MNIST
import matplotlib.pyplot as plt
%matplotlib inline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.autograd import Variable
from sklearn.metrics import confusion_matrix
import pandas as pd
```

Setup

Download and Process Data

In [0]:
```
# Download the dataset.
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-images-idx3-ubyte.gz -o /tmp/train-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/tra
in-labels-idx1-ubyte.gz -o /tmp/train-labels-idx1-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-images-idx3-ubyte.gz -o /tmp/t10k-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10
k-labels-idx1-ubyte.gz -o /tmp/t10k-labels-idx1-ubyte.gz')
pass
```

In [0]:
```
# load the dataset with shape (*, 784)
mnistdata = MNIST("/tmp")
mnistdata.gz = True
trainxs_raw, trainys_raw = mnistdata.load_training()
testxs_raw, testys_raw = mnistdata.load_testing()

# reshape data into square image (*, 1, 28, 28)
trainxs_square  = np.array(trainxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
trainys_square  = np.array(trainys_raw, dtype=np.long)
testxs          = np.array(testxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
testys          = np.array(testys_raw)

# split training dataset
trainxs = trainxs_square[:50000]
trainys = trainys_square[:50000]
validxs = trainxs_square[50000:]
validys = trainys_square[50000:]
```

In [0]:
```
# process into FashionMNist 1 and Fashon MNist 2

def one_hot_encode(ys, labels):
  one_hot = np.zeros((ys.shape[0], len(labels)))
  position_dict = {labels[i]: i for i in range(len(labels))}
  for label in position_dict.keys():
    one_hot[ys == label, position_dict[label]] = 1.
  return one_hot

def filter_for_labels(trainxs, trainys, validxs, validys, testxs, testys, labels):
    # FashinoMNist 1
    train_ix = np.isin(trainys, labels)
    trainxs_ = trainxs[train_ix]
    trainys_ = trainys[train_ix]

    val_ix = np.isin(validys, labels)
    validxs_ = validxs[val_ix]
    validys_ = validys[val_ix]

    test_ix = np.isin(testys, labels)
    testxs_ = testxs[test_ix]
    testys_ = testys[test_ix]

    return trainxs_, trainys_, validxs_, validys_, testxs_, testys_

labels_1 = [0, 1, 4, 5, 8]
labels_2 = [x for x in list(range(10)) if x not in labels_1]
labels_1_to_position = {labels_1[i]: i for i in range(len(labels_1))}
labels_2_to_position = {labels_2[i]: i for i in range(len(labels_2))}
position_to_labels_1 = {v: k for k, v in labels_1_to_position.items()}
position_to_labels_2 = {v: k for k, v in labels_2_to_position.items()}

def labels_to_positions(labels, labels_to_positions):
  positions = np.full(len(labels), -1)
  for label, position in labels_to_positions.items():
   ix = (labels == label).nonzero()
   positions[ix] = position
  if (positions < 0).any():
   print('WARNING! invalid label index!')
  return torch.from_numpy(positions)

def accuracy(model, X, Y):
  outputs = model(X)
  _, prediction = torch.max(outputs, 1)
  if torch.cuda.is_available():
      prediction = prediction.cpu()
  return sum((Y - prediction.numpy())==0) / len(Y) * 100


# FashinoMNist1
trainxs_1, trainys_1, validxs_1, validys_1, testxs_1, testys_1 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_1
)

#FashionMNist2
trainxs_2, trainys_2, validxs_2, validys_2, testxs_2, testys_2 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_2
)

trainxs_comb = np.concatenate([trainxs_1.copy(), trainxs_2.copy(), validxs_2.copy
(), testxs_2.copy()])
trainys_comb = np.concatenate([trainys_1.copy(), trainys_2.copy(), validys_2.copy
(), testys_2.copy()])
shuffled_ix = list(range(trainxs_comb.shape[0]))
```

In [0]:
```
# dataset examples
fig=plt.figure(figsize=(20, 5))
columns = 20
rows = 5
for i in range(columns*rows):
    fig.add_subplot(rows, columns, i+1)
    plt.axis('off')
    plt.imshow(trainxs[i, 0], cmap='binary')
plt.show()
```

Structure Dataset

In [0]:
```
class DatasetConstructor(Dataset):

    def __init__(self, X, Y):
        self.X = torch.from_numpy(X)
        self.Y = torch.from_numpy(Y)
        self.len = X.shape[0]

    def __getitem__(self, index):
        return self.X[index], self.Y[index]

    def __len__(self):
        return self.len

BATCHSIZE = 32

train_comb_data = DatasetConstructor(trainxs_comb, trainys_comb)
train_1_data = DatasetConstructor(trainxs_1, trainys_1)
valid_1_data = DatasetConstructor(validxs_1, validys_1)
test_1_data  = DatasetConstructor(testxs_1, testys_1)

train_comb_loader =  DataLoader(dataset=train_comb_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
train_1_loader  = DataLoader(dataset=train_1_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
valid_1_loader  = DataLoader(dataset=valid_1_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
test_1_loader   = DataLoader(dataset=test_1_data, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)

# tensor dataset for dataset loss computation
trainxs_comb_tensor = torch.from_numpy(trainxs_comb)
trainys_comb_tensor = torch.from_numpy(trainys_comb)

trainxs_1_tensor = torch.from_numpy(trainxs_1)
trainys_1_tensor = torch.from_numpy(trainys_1)

validxs_1_tensor = torch.from_numpy(validxs_1)
validys_1_tensor = torch.from_numpy(validys_1)

testxs_1_tensor  = torch.from_numpy(testxs_1)
testys_1_tensor  = torch.from_numpy(testys_1)
```

Exercise

1. Implement an autoencoder with mean squared error loss for the Fashion-MNIST-1 and Fashion-MNIST-2 data.

In [0]:
```
class DenseAutoEncoder(nn.Module):
    def __init__(self):
        super(DenseAutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 32),
            nn.ReLU(True),
            nn.Linear(32, 5),
            nn.Sigmoid()
        )
        self.decoder = nn.Sequential(
            nn.Linear(5, 256),
            nn.ReLU(True),
            nn.Linear(256, 128),
            nn.ReLU(True),
            nn.Linear(128, 28*28),
            nn.ReLU(True)
        )

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.encoder(x)
        x = self.decoder(x)
        x = x.view(-1, 1, 28, 28)
        return x
```

2. Train your model to convergence on the combined training, validation, and test set of Fashion-MNIST-2 and training set of Fashion-MNIST-1 using an optimisation algorithm of your choice.

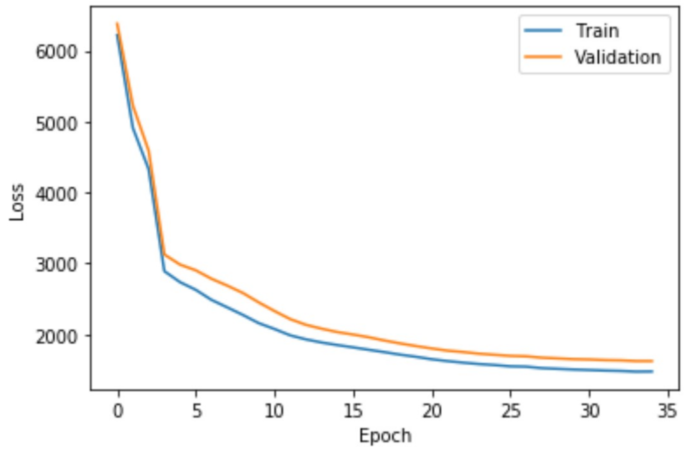
Note, for now we'll just use the validation set from FashionMNist 1... this will probably mean we'll get a significant difference in training and validation error since they come from different distributions, but at least it gives us an idea of whether or not our encoder is generalising beyond training data.

In [0]:
```
# define model and attach to GPU if available
autoencoder = DenseAutoEncoder()
if torch.cuda.is_available():
    autoencoder.cuda()

# define optimizer and mean-squared error loss
loss = nn.MSELoss()
optimizer = torch.optim.Adam(autoencoder.parameters(), lr=0.0001,
                             weight_decay=1e-5)

# train model until convergence
train_loss = []
valid_loss = []
epochs = 35
live = True

for epoch in range(epochs):  # loop over the dataset multiple times
    for i, data in enumerate(train_comb_loader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        if torch.cuda.is_available():
            inputs, labels = inputs.cuda(), labels.cuda()
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = autoencoder(inputs)
        loss_val = loss(outputs, inputs)
        loss_val.backward()
        optimizer.step()

    if torch.cuda.is_available():
        train_loss.append(loss(autoencoder(trainxs_comb_tensor.cuda()), trainxs_com
b_tensor.cuda()).item())
        valid_loss.append(loss(autoencoder(validxs_1_tensor.cuda()), validxs_1_tens
or.cuda()).item())
    else:
        train_loss.append(loss(autoencoder(trainxs_comb_tensor), trainxs_comb_tenso
r).item())
        valid_loss.append(loss(autoencoder(validxs_1_tensor), validxs_1_tensor).ite
m())

    if live:
      print('Training Loss at epoch {}: {}'.format(epoch, train_loss[-1]))
      print('Validation Loss at epoch {}: {}'.format(epoch, valid_loss[-1]))


print('Finished Training')

# save modelb
PATH = './autoenc_net.pth'
torch.save(autoencoder.state_dict(), PATH)
print("Model Saved")
```

```
Training Loss at epoch 0: 6219.47705078125
Validation Loss at epoch 0: 6380.5546875
Training Loss at epoch 1: 4910.87158203125
Validation Loss at epoch 1: 5224.82666015625
Training Loss at epoch 2: 4330.587890625
Validation Loss at epoch 2: 4590.93310546875
Training Loss at epoch 3: 2891.427978515625
Validation Loss at epoch 3: 3126.725830078125
Training Loss at epoch 4: 2736.500244140625
Validation Loss at epoch 4: 2984.04052734375
Training Loss at epoch 5: 2628.14990234375
Validation Loss at epoch 5: 2903.4541015625
Training Loss at epoch 6: 2486.513916015625
Validation Loss at epoch 6: 2784.145263671875
Training Loss at epoch 7: 2381.31982421875
Validation Loss at epoch 7: 2688.048583984375
Training Loss at epoch 8: 2273.89013671875
Validation Loss at epoch 8: 2583.37060546875
Training Loss at epoch 9: 2158.9208984375
Validation Loss at epoch 9: 2451.37255859375
Training Loss at epoch 10: 2076.306396484375
Validation Loss at epoch 10: 2330.00146484375
Training Loss at epoch 11: 1987.0321044921875
Validation Loss at epoch 11: 2215.91259765625
Training Loss at epoch 12: 1930.07421875
Validation Loss at epoch 12: 2135.132080078125
Training Loss at epoch 13: 1885.946533203125
Validation Loss at epoch 13: 2080.3447265625
Training Loss at epoch 14: 1849.8035888671875
Validation Loss at epoch 14: 2032.620361328125
Training Loss at epoch 15: 1817.1922607421875
Validation Loss at epoch 15: 1997.885498046875
Training Loss at epoch 16: 1783.1783447265625
Validation Loss at epoch 16: 1959.6842041015625
Training Loss at epoch 17: 1750.1077880859375
Validation Loss at epoch 17: 1913.491943359375
Training Loss at epoch 18: 1714.220947265625
Validation Loss at epoch 18: 1872.5252685546875
Training Loss at epoch 19: 1684.3289794921875
Validation Loss at epoch 19: 1835.5938720703125
Training Loss at epoch 20: 1649.8333740234375
Validation Loss at epoch 20: 1801.79736328125
Training Loss at epoch 21: 1623.650634765625
Validation Loss at epoch 21: 1770.7205810546875
Training Loss at epoch 22: 1601.6861572265625
Validation Loss at epoch 22: 1751.5670166015625
Training Loss at epoch 23: 1582.3267822265625
Validation Loss at epoch 23: 1728.083251953125
Training Loss at epoch 24: 1568.0657958984375
Validation Loss at epoch 24: 1712.2486572265625
Training Loss at epoch 25: 1550.197509765625
Validation Loss at epoch 25: 1697.0211181640625
Training Loss at epoch 26: 1545.279541015625
Validation Loss at epoch 26: 1690.9493408203125
Training Loss at epoch 27: 1524.298583984375
Validation Loss at epoch 27: 1671.232177734375
Training Loss at epoch 28: 1514.6461181640625
Validation Loss at epoch 28: 1661.538330078125
Training Loss at epoch 29: 1504.8077392578125
Validation Loss at epoch 29: 1650.8875732421875
Training Loss at epoch 30: 1498.033447265625
Validation Loss at epoch 30: 1646.359130859375
Training Loss at epoch 31: 1490.2166748046875
Validation Loss at epoch 31: 1637.59912109375
```

Let's plot a graph to double check that the model has indeed converged:

In [0]:
```
fig = plt.figure()
plt.plot(train_loss)
plt.plot(valid_loss)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(['Train', 'Validation']);
```



See what the encoder looks like:

In [0]:
```
 # load model
temp_autoencoder = autoencoder.cpu()
#temp_autoencoder.load_state_dict(torch.load(PATH))
outputs = temp_autoencoder(torch.from_numpy(testxs[:10]))

# plot decoded encodings
fig, axes = plt.subplots(2, 10, figsize=(12, 3))
for i in range(10):
  axes[0, i].imshow(testxs[i, 0], cmap='binary')
  axes[0, i].axis('off')
  axes[1, i].imshow(outputs[i, 0].data, cmap='binary')
  axes[1, i].axis('off')
```

3. Implement a multi-class, multi-layer perceptron with cross-entropy loss for the FashionMNIST-1 data, which shares the same structure as the encoder of your autoencoder.

In [0]:
```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.encoder = DenseAutoEncoder().encoder

    def load_hiddens(self, state_dict):
      self.encoder.load_state_dict(state_dict)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.encoder(x)
        return x
```

4. Compare using random weights to those from your autoencoder to initialise the multi-layer perceptron by plotting the training and validation loss for both options when you use 5%, 10%, . . . , 100% of the available Fashion-MNIST-1 training data to train your model. Is there a point where one initialisation option is superior to the other? Is one option always superior to the other?

In [0]:
```
def build_dataset(xs, ys, percentage=100):
  m = xs.shape[0]
  assert(m == ys.shape[0])
  cutoff = int((percentage/100)*m)

  permutation_ix = np.random.permutation(list(range(m)))
  xs_perm = xs[permutation_ix]
  ys_perm = ys[permutation_ix]

  return DatasetConstructor(xs_perm[:cutoff], ys_perm[:cutoff])

def build_dataloader(xs, ys, percentage=100, batch_size=BATCHSIZE):
  dataset = build_dataset(xs, ys, percentage)
  return DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=2)
```

In [0]:
```
validys_1_position_tensor = labels_to_positions(validys_1_tensor, labels_1_to_posit
ion)
```

In [0]:
```
from tqdm import tqdm

percentages = list(range(5, 105, 5))
random = {percentage: {'valid': [], 'train': []} for percentage in percentages}
epochs = 20
live=True

# random initialisations
for percent in tqdm(random.keys()):
  # build training data
  train_perc_loader = build_dataloader(trainxs_1, trainys_1, percentage=percent)
  trainxs_perc = train_perc_loader.dataset.X
  trainys_perc = labels_to_positions(train_perc_loader.dataset.Y, labels_1_to_posit
ion)

  # define model, loss and optimiser
  mlp = MLP()
  if torch.cuda.is_available():
    mlp.cuda()
  loss = nn.CrossEntropyLoss()
  optimizer = torch.optim.Adam(mlp.parameters(), lr=0.00001,
                               weight_decay=1e-5)

  # train model
  for epoch in range(epochs):  # loop over the dataset multiple times
    for i, data in enumerate(train_perc_loader, 0):
      # get the inputs; data is a list of [inputs, labels]
      inputs, labels = data
      labels = labels_to_positions(labels, labels_1_to_position)
      inputs = inputs

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()
      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = mlp(inputs)
      loss_val = loss(outputs, labels)
      loss_val.backward()
      optimizer.step()

    if torch.cuda.is_available():
      random[percent]['train'].append(loss(mlp(trainxs_perc.cuda()), trainys_perc.c
uda()).item())
      random[percent]['valid'].append(loss(mlp(validxs_1_tensor.cuda()), validys_1_
position_tensor.cuda()).item())
    else:
      random[percent]['train'].append(loss(mlp(trainxs_perc), trainys_perc).item())
      random[percent]['valid'].append(loss(mlp(validxs_1_tensor), validys_1_positio
n_tensor).item())


  if live:
    print(random[percent]['train'])
    print(accuracy(mlp, validxs_1_tensor.cuda(), validys_1))
  print('Finished Training for ', percent, '% data.')
```

```
  5%|▌          | 1/20 [00:06<01:54,  6.01s/it]

[1.4818170070648193, 1.366442084312439, 1.2828389406204224, 1.2301939725875854,
1.1922353506088257, 1.164351224899292, 1.1409800052642822, 1.11957848072052, 1.0
93044400215149, 1.0741167068481445, 1.0610089302062988, 1.0505481958389282, 1.04
21135425567627, 1.0346524715423584, 1.0280171632766724, 1.022196650505066, 1.016
5799856185913, 1.0111470222473145, 1.0054073333740234, 0.9997521638870239]
37.43283582089552
Finished Training for  5 % data.

 10%|█          | 2/20 [00:15<02:07,  7.06s/it]

[1.4245538711547852, 1.3588385581970215, 1.3288882970809937, 1.3097978830337524,
1.2973588705062866, 1.2861660718917847, 1.2730361223220825, 1.2613871097564697,
1.2463886737823486, 1.2385334968566895, 1.2332426309585571, 1.2252604961395264,
1.1543720960617065, 1.1323907375335693, 1.1149091720581055, 1.1030094623565674,
1.0964423418045044, 1.0910106897354126, 1.0872104167938232, 1.0830764770507812]
19.30348258706468
Finished Training for  10 % data.

 15%|█▌         | 3/20 [00:29<02:33,  9.05s/it]

[1.3741801977157593, 1.2113540172576904, 1.161368489265442, 1.1352205276489258,
1.118154764175415, 1.1058639287948608, 1.0979291200637817, 1.0917046070098877,
1.0869626998901367, 1.083712100982666, 1.0805318355560303, 1.0780162811279297,
1.0759272575378418, 1.0742217302322388, 1.0726840496063232, 1.071166753768921,
1.0703444480895996, 1.0690630674362183, 1.0679426193237305, 1.0669691562652588]
47.20398009950249
Finished Training for  15 % data.

 20%|██         | 4/20 [00:46<03:03, 11.48s/it]

[1.2246882915496826, 1.081337332725525, 1.0288392305374146, 0.9960038065910339,
0.9780386090278625, 0.9659217596054077, 0.9586060643196106, 0.9530305862426758,
0.9488063454627991, 0.945923388004303, 0.942748486995697, 0.9408616423606873, 0.
9388712048530579, 0.9366848468780518, 0.9355881214141846, 0.9337881803512573, 0.
9324766993522644, 0.9312095046043396, 0.9303257465362549, 0.9294505715370178]
38.766169154228855
Finished Training for  20 % data.

 25%|██▌        | 5/20 [01:07<03:36, 14.41s/it]

[1.1585240364074707, 1.0460000038146973, 1.0016493797302246, 0.978823721408844,
0.9651063084602356, 0.9569585919380188, 0.9498911499977112, 0.9454324245452881,
0.9416418075561523, 0.9389358162879944, 0.936678946018219, 0.9352486729621887,
0.9330644011497498, 0.9316138029098511, 0.9304598569869995, 0.9290786385536194,
0.9283915162086487, 0.9271453022956848, 0.9262374043464661, 0.925452470779419]
38.96517412935324
Finished Training for  25 % data.

 30%|███        | 6/20 [01:31<04:03, 17.40s/it]

[1.199047565460205, 1.1338335275650024, 1.1087809801101685, 1.0973953008651733,
1.0900465250015259, 1.0858231782913208, 1.0818842649459839, 1.0790809392929077,
1.0769487619400024, 1.0756467580795288, 1.0733493566513062, 1.0717285871505737,
1.0703519582748413, 1.070251703262329, 1.068509578704834, 1.0680458545684814, 1.
0667909383773804, 1.0665152072906494, 1.0656174421310425, 1.0649595260620117]
44.31840796019901
Finished Training for  30 % data.

 35%|███▌       | 7/20 [01:59<04:26, 20.51s/it]
```

```
[1.0774405002593994, 0.9999778866767883, 0.972917914390564, 0.9608140587806702,
0.9526200294494629, 0.9479729533195496, 0.9438551068305969, 0.9403189420700073,
0.9379003047943115, 0.9357064366340637, 0.9331540465354919, 0.932739794254303,
0.93024080991745, 0.9292702078819275, 0.928111359596252, 0.9272171258926392, 0.
9260481595993042, 0.925057590007782, 0.9241029024124146, 0.9243932366371155]
38.746268656716424
Finished Training for  35 % data.

 40%|████      | 8/20 [02:31<04:45, 23.80s/it]

[1.1579022407531738, 0.9842104911804199, 0.9645397067070007, 0.9548328518867493,
0.9486057162284851, 0.9447070360183716, 0.9409112334251404, 0.9386374950408936,
0.9358670115470886, 0.9342265129089355, 0.9323087930679321, 0.9313400387763977,
0.9297012686729431, 0.9291222095489502, 0.9273682832717896, 0.9264588952064514,
0.926257312297821, 0.9246441721916199, 0.9242343902587891, 0.9241169691085815]
39.04477611940298
Finished Training for  40 % data.

 45%|████▌     | 9/20 [03:05<04:57, 27.02s/it]

[1.142991542816162, 1.0987045764923096, 1.0869338512420654, 1.0809235572814941,
1.0763338804244995, 1.0730974674224854, 1.070910930633545, 1.069291114807129, 1.
067857265472412, 1.0667963027954102, 1.0656782388687134, 1.0647355318069458, 1.0
636601448059082, 1.0628480911254883, 1.0626964569091797, 1.061662197113037, 1.06
10688924789429, 1.0606231689453125, 1.0601580142974854, 1.060089349746704]
47.66169154228856
Finished Training for  45 % data.

 50%|█████     | 10/20 [03:42<04:58, 29.88s/it]

[1.0622299909591675, 0.9914138317108154, 0.9638216495513916, 0.9514317512512207,
0.9453655481338501, 0.9404813647270203, 0.9370772242546082, 0.9363911747932434,
0.9325776100158691, 0.9308164119720459, 0.9292705655097961, 0.9280255436897278,
0.9272327423095703, 0.9262974858283997, 0.9253842830657959, 0.924531340599060l,
0.9232660531997681, 0.9226965308189392, 0.9219794273376465, 0.9215584993362427]
39.06467661691542
Finished Training for  50 % data.

 55%|█████▌    | 11/20 [04:21<04:55, 32.79s/it]

[1.0587376356124878, 0.9782001972198486, 0.9574517011642456, 0.9484888911247253,
0.9427201747894287, 0.93876051902771, 0.9362161755561829, 0.9339087009429932, 0.
9319803714752197, 0.9300916790962219, 0.9290516972541809, 0.9276663661003113, 0.
9267885684967041, 0.9258579611778259, 0.9252893924713135, 0.9242820739746094, 0.
9233736991882324, 0.922741174697876, 0.9227566719055176, 0.9216471314430237]
39.20398009950249
Finished Training for  55 % data.

 60%|██████    | 12/20 [05:07<04:52, 36.51s/it]

[1.0530837774276733, 0.9703307747840881, 0.9513116478919983, 0.9431861042976379,
0.938625156879425, 0.9352293014526367, 0.9328870177268982, 0.9309885501861572,
0.929473876953125, 0.9280543327331543, 0.9276003837585449, 0.9264971613883972,
0.9259674549102783, 0.924985945224762, 0.9240211844444275, 0.9228757619857788,
0.9223050475120544, 0.9215830564498901, 0.9214226603507996, 0.9205580353736877]
39.08457711442786
Finished Training for  60 % data.

 65%|██████▌   | 13/20 [05:54<04:39, 39.92s/it]
```

```
[1.0025368928909302, 0.9609987735748291, 0.9483929872512817, 0.941531240940094,
0.9369444251060486, 0.9336274862289429, 0.931174099445343, 0.9291001558303833,
0.9275785088539124, 0.9265837669372559, 0.9257522821426392, 0.9243525862693787,
0.923194825649261 5, 0.9224235415458679, 0.9228016138076782, 0.9211103320121765,
0.9208662509918213, 0.9200239777565002, 0.919617772102356, 0.919461727142334]
39.08457711442786
Finished Training for  65 % data.

 70%|███████    | 14/20 [06:48<04:24, 44.05s/it]

[1.0417519807815552, 0.9998714327812195, 0.9653394818305969, 0.9476131796836853,
0.9406654238700867, 0.9366224408149719, 0.934053361415863, 0.9312599897384644,
0.9293537139892578, 0.9279443025588989, 0.9266924858093262, 0.925888180732727,
0.9244229197502136, 0.9239820837974548, 0.9236012697219849, 0.9221677184104919,
0.9217789173126221, 0.9209482669830322, 0.9204067587852478, 0.9200099110603333]
39.02487562189055
Finished Training for  70 % data.

 75%|████████   | 15/20 [07:45<03:58, 47.76s/it]

[1.182664394378662, 1.1367888450622559, 1.1236435174942017, 1.1194480657577515,
1.115181565284729, 1.1129928827285767, 1.110497236251831, 1.1089268922805786, 1.
1075717210769653, 1.1066725254058838, 1.1057155132293701, 0.9335666298866272, 0.
9281078577041626, 0.9259046912193298, 0.924822986125946, 0.9234029054641724, 0.9
22793447971344, 0.9218996167182922, 0.9220394492149353, 0.9205272793769836]
39.223880597014926
Finished Training for  75 % data.

 80%|████████   | 16/20 [08:42<03:22, 50.74s/it]

[1.1153676509857178, 1.0874890089035034, 0.9490423798561096, 0.9409382939338684,
0.9367676377296448, 0.93300861120224, 0.930628776550293, 0.9289974570274353, 0.9
275793433189392, 0.9264881014823914, 0.9253003001213074, 0.9241728186607361, 0.9
237836599349976, 0.9229421615600586, 0.9227988719940186, 0.922116756439209, 0.92
10649132728577, 0.9211719632148743, 0.9199490547180176, 0.9195889234542847]
39.18407960199005
Finished Training for  80 % data.

 85%|█████████  | 17/20 [09:42<02:40, 53.43s/it]

[1.2225563526153564, 1.2083488702774048, 1.203543782234192, 1.2006281614303589,
1.199191927909851, 1.197414517402649, 1.1964260339736938, 1.1952180862426758, 1.
1942155361175537, 1.1929388046264648, 1.0674306154251099, 1.0633127689361572, 1.
0614241361618042, 1.060451865196228, 1.0597174167633057, 1.058690071105957, 1.05
80432415008545, 1.0575907230377197, 1.0569525957107544, 1.0566883087158203]
43.64179104477612
Finished Training for  85 % data.

 90%|█████████ | 18/20 [10:49<01:54, 57.46s/it]

[1.1036286354064941, 0.987105131149292, 0.944254457950592, 0.9369871616363525,
0.9330489039421082, 0.9310418367385864, 0.9285155534744263, 0.9269367456436157,
0.9258151650428772, 0.9245151281356812, 0.9233529567718506, 0.9228511452674866,
0.9220007658004761, 0.9215001463890076, 0.9206239581108093, 0.9200531840324402,
0.920187771320343, 0.9194954633712769, 0.9186162948608398, 0.9183179140090942]
39.06467661691542
Finished Training for  90 % data.

 95%|██████████| 19/20 [12:01<01:01, 61.78s/it]
```

```
[0.9795790910720825, 0.950786292552948, 0.9411289095878601, 0.9354758262634277,
0.9314044713973999, 0.9284194707870483, 0.9267678260803223, 0.9250442981719971,
0.9245555996894836, 0.9227747321128845, 0.922189474105835, 0.9209747314453125,
0.92033451795578, 0.9197425246238708, 0.9193764328956604, 0.9188597798347473, 0.
9189032316207886, 0.9178138375282288, 0.9176976680755615, 0.917281448841095]
39.20398009950249
Finished Training for  95 % data.
```

```
100%|██████████| 20/20 [13:16<00:00, 65.80s/it]
```

```
[0.983206570148468, 0.9499852061271667, 0.9396902918815613, 0.9349431395530701,
0.9327477216720581, 0.9291402101516724, 0.9271497130393982, 0.9258050918579102,
0.9248872995376587, 0.9235685467720032, 0.9230046272277832, 0.9225431680679321,
0.9215736985206604, 0.9205540418624878, 0.9200090765953064, 0.9198775887489319,
0.9188680648803711, 0.9187155365943909, 0.91805499792099, 0.9176715016365051]
39.2636815920398
Finished Training for  100 % data.
```

In [0]:
```
epochs = 20
live=True

percentages = list(range(5, 105, 5))
pretrained = {percentage: {'valid': [], 'train': []} for percentage in percentages}

# encoded initialisations
for percent in tqdm(pretrained.keys()):
  # build training data
  train_perc_loader = build_dataloader(trainxs_1, trainys_1, percentage=percent)
  trainxs_perc = train_perc_loader.dataset.X
  trainys_perc = labels_to_positions(train_perc_loader.dataset.Y, labels_1_to_posit
ion)

  # define model, loss and optimiser
  mlp = MLP()
  # initialise model weights
  mlp.load_hiddens(autoencoder.encoder.state_dict())

  if torch.cuda.is_available():
    mlp.cuda()
  loss = nn.CrossEntropyLoss()
  optimizer = torch.optim.Adam(mlp.parameters(), lr=0.0001,
                               weight_decay=1e-5)

  # train model
  for epoch in range(epochs):  # loop over the dataset multiple times
    for i, data in enumerate(train_perc_loader, 0):
      # get the inputs; data is a list of [inputs, labels]
      inputs, labels = data
      labels = labels_to_positions(labels, labels_1_to_position)
      inputs = inputs

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()
      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = mlp(inputs)
      loss_val = loss(outputs, labels)
      loss_val.backward()
      optimizer.step()

    if torch.cuda.is_available():
      pretrained[percent]['train'].append(loss(mlp(trainxs_perc.cuda()), trainys_pe
rc.cuda()).item())
      pretrained[percent]['valid'].append(loss(mlp(validxs_1_tensor.cuda()), validy
s_1_position_tensor.cuda()).item())
    else:
      pretrained[percent]['train'].append(loss(mlp(trainxs_perc), trainys_perc).ite
m())
      pretrained[percent]['valid'].append(loss(mlp(validxs_1_tensor), validys_1_pos
ition_tensor).item())

  if live:
    print(pretrained[percent]['train'])
    print(accuracy(mlp, validxs_1_tensor.cuda(), validys_1))
  print('Finished Training for ', percent, '% data.')
```

```
  5%|▌         | 1/20 [00:06<01:55,  6.07s/it]

[1.1763304471969604, 1.0296040773391724, 0.9838058352470398, 0.961625337600708,
0.9502491354942322, 0.9444736838340759, 0.9399560689926147, 0.9336432218551636,
0.9320246577262878, 0.9277067184448242, 0.9249801635742188, 0.9233349561691284,
0.9233719110488892, 0.9206917881965637, 0.9191098213195801, 0.9187025427818298,
0.9174138307571411, 0.9167070388793945, 0.9170057773590088, 0.9150090217590332]
38.646766169154226
Finished Training for  5 % data.

 10%|█         | 2/20 [00:15<02:08,  7.14s/it]

[1.0280848741531372, 0.9679586887359619, 0.9518606066703796, 0.9408490657806396,
0.9348258972167969, 0.9309307932853699, 0.9269300699234009, 0.925048828125, 0.92
18398928642273, 0.919793426990509, 0.9218829274177551, 0.9176096320152283, 0.916
4831638336182, 0.9161227941513062, 0.9151527881622314, 0.9143863320350647, 0.913
9128923416138, 0.9156655073165894, 0.9131808876991272, 0.9131532311439514]
39.08457711442786
Finished Training for  10 % data.

 15%|█▌        | 3/20 [00:28<02:32,  8.99s/it]

[0.983338475227356, 0.9489954710006714, 0.9382382035255432, 0.9317742586135864,
0.9280170798301697, 0.9231098294258118, 0.920979380607605, 0.9191848635673523,
0.9179655313491821, 0.917770266532898, 0.9154490232467651, 0.9158105254173279,
0.9145792126655579, 0.9132707715034485, 0.9128679037094116, 0.9116482734680176,
0.9118956327438354, 0.9114851951599121, 0.9113416075706482, 0.9156243801116943]
38.70646766169154
Finished Training for  15 % data.

 20%|██        | 4/20 [00:45<02:59, 11.23s/it]

[0.9660235643386841, 0.9434491991996765, 0.9349028468132019, 0.9284136891365051,
0.9265075325965881, 0.9215176105499268, 0.9191381931304932, 0.9177894592285156,
0.9169210195541382, 0.9157286882400513, 0.9162864089012146, 0.9141250848770142,
0.9130286574363708, 0.913609504699707, 0.9125558137893677, 0.913615882396698, 0.
9129453897476196, 0.9119774103164673, 0.9113174676895142, 0.9109140634536743]
39.02487562189055
Finished Training for  20 % data.

 25%|██▌       | 5/20 [01:05<03:26, 13.79s/it]

[0.9579474329948425, 0.9403177499771118, 0.9342588782310486, 0.9317845106124878,
0.92661452293396, 0.9267374873161316, 0.9231225848197937, 0.9208466410636902, 0.
9195195436477661, 0.9199889898300171, 0.9178950190544128, 0.9173194169998169, 0.
9165301322937012, 0.9194751381874084, 0.9155354499816895, 0.9150591492652893, 0.
9163560271263123, 0.914354681968689, 0.9141566753387451, 0.9132670164108276]
39.223880597014926
Finished Training for  25 % data.

 30%|███       | 6/20 [01:28<03:52, 16.61s/it]

[0.9538162350654602, 0.9348653554916382, 0.9296329617500305, 0.9263929724693298,
0.9238154888153076, 0.9231377243995667, 0.9195526838302612, 0.9176364541053772,
0.9174533486366272, 0.9159482717514038, 0.9163747429847717, 0.914635956287384,
0.9148766398429871, 0.9140501022338867, 0.9139785170555115, 0.9126559495925903,
0.9123356938362122, 0.9135808944702148, 0.9124054312705994, 0.9127068519592285]
39.16417910447761
Finished Training for  30 % data.

 35%|███▌      | 7/20 [01:54<04:13, 19.46s/it]
```

```
[0.9480588436126709, 0.9367730617523193, 0.9305466413497925, 0.9255104064941406,
0.9234601259231567, 0.9238159656524658, 0.9199320077896118, 0.918744683265686,
0.9224665760993958, 0.917785108089447, 0.9179590940475464, 0.9175682067871094,
0.9156380891799927, 0.9148933291435242, 0.9153395891189575, 0.9132573008537292,
0.9131511449813843, 0.9133646488189697, 0.9137815237045288, 0.9122528433799744]
39.243781094527364
Finished Training for  35 % data.

 40%|████      | 8/20 [02:23<04:26, 22.24s/it]

[0.9480270147323608, 0.9331294894218445, 0.9271416068077087, 0.9248270988464355,
0.9233781695365906, 0.9209531545639038, 0.921410083770752, 0.9182369709014893,
0.919065535068512, 0.9175821542739868, 0.920561671257019, 0.9159466028213501, 0.
9165195226669312, 0.914125382900238, 0.9138966798782349, 0.9154422879219055, 0.9
136413931846619, 0.9130584597587585, 0.9132223129272461, 0.9126947522163391]
39.34328358208955
Finished Training for  40 % data.

 45%|████▌     | 9/20 [02:55<04:36, 25.10s/it]

[0.9428200721740723, 0.9320834875106812, 0.9271494746208191, 0.9229021668434143,
0.9226809740066528, 0.9201131463050842, 0.9191702008247375, 0.9189566969871521,
0.9180290699005127, 0.9166731834411621, 0.9188867211341858, 0.9207478165626526,
0.916864812374115, 0.9161069393157959, 0.9162521362304688, 0.9144109487533569,
0.915034830570221, 0.91322261095047, 0.9131678938865662, 0.9125161170959473]
39.08457711442786
Finished Training for  45 % data.

 50%|█████     | 10/20 [03:33<04:50, 29.04s/it]

[0.940662145614624, 0.9293780326843262, 0.9245340824127197, 0.9218980073928833,
0.9224313497543335, 0.9202002882957458, 0.917127251625061, 0.9176527857780457,
0.9178153276443481, 0.9157586097717285, 0.9170543551445007, 0.9166528582572937,
0.914161205291748, 0.9143063426017761, 0.9158428907394409, 0.9128208160400391,
0.9130435585975647, 0.9130557775497437, 0.9121938943862915, 0.9124591946601868]
39.32338308457711
Finished Training for  50 % data.

 55%|█████▌    | 11/20 [04:13<04:52, 32.48s/it]

[0.9424868822097778, 0.9280375838279724, 0.9243621230125427, 0.9257498979568481,
0.920045018196106, 0.9186038970947266, 0.9202559590339661, 0.9171779751777649,
0.9161962270736694, 0.9154295921325684, 0.9157094359397888, 0.9142704010009766,
0.9148361086845398, 0.9134527444839478, 0.9137014746665955, 0.91334468126297, 0.
91328364610672, 0.912875235080719, 0.9114670753479004, 0.9115496873855591]
39.28358208955224
Finished Training for  55 % data.

 60%|██████    | 12/20 [04:59<04:50, 36.34s/it]

[0.9371150732040405, 0.926276683807373, 0.923935055732727, 0.9206237196922302,
0.9191631078720093, 0.9190729260444641, 0.918658971786499, 0.9177441596984863,
0.9161028265953064, 0.9153242707252502, 0.9159386157989502, 0.9145819544792175,
0.9170199632644653, 0.913882851600647, 0.9138236045837402, 0.9154949188232422,
0.9138539433479309, 0.9143757224082947, 0.9135964512825012, 0.9137721657752991]
39.08457711442786
Finished Training for  60 % data.

 65%|██████▌   | 13/20 [05:47<04:40, 40.08s/it]
```

[0.9379402995109558, 0.9281522035598755, 0.9233096241950989, 0.9214868545532227, 0.9207853674888611, 0.9198504686355591, 0.9189116358757019, 0.9198563098907471, 0.9168712496757507, 0.9156304001808167, 0.9151895046234131, 0.9144683480262756, 0.9139237403869629, 0.9146284461021423, 0.9131289124488831, 0.9133589863777161, 0.9131996035575867, 0.9121379256248474, 0.9149847030639648, 0.9115374088287354]
39.28358208955224
Finished Training for 65 % data.

70%|███████ | 14/20 [06:40<04:22, 43.73s/it]

[0.9363995790481567, 0.928364634513855, 0.9234556555747986, 0.9225414991378784, 0.9201731085777283, 0.9211522340774536, 0.9180041551589966, 0.918961763381958, 0.9201206564903259, 0.918133556842804, 0.9151772856712341, 0.9160847663879395, 0.9150773882865906, 0.915107250213623, 0.9169347286224365, 0.9153790473937988, 0.9164083003997803, 0.9132139086723328, 0.9131060242652893, 0.9138321876525879]
39.1044776119403
Finished Training for 70 % data.

75%|███████▌ | 15/20 [07:36<03:57, 47.46s/it]

[0.9359508156776428, 0.9274254441261292, 0.9253124594688416, 0.9216245412826538, 0.9237291812896729, 0.9209617972373962, 0.9194897413253784, 0.9181182980537415, 0.9160316586494446, 0.915558934211731, 0.9166755676269531, 0.9165722131729126, 0.9146040081977844, 0.9144213795661926, 0.9144569635391235, 0.9133399128913879, 0.9135453104972839, 0.9117933511734009, 0.9127694368362427, 0.9117608070373535]
39.482587064676615
Finished Training for 75 % data.

80%|████████ | 16/20 [08:36<03:25, 51.29s/it]

[0.9341229200363159, 0.9272401332855225, 0.9233902096748352, 0.9227387309074402, 0.9204570651054382, 0.9203280210494995, 0.9241381883621216, 0.9188626408576965, 0.916521430015564, 0.9152454137802124, 0.9164524674415588, 0.9144638776779175, 0.9164292216300964, 0.9146885275840759, 0.9130775332450867, 0.9124212265014648, 0.9125366806983948, 0.9125062823295593, 0.9125628471374512, 0.9118631482124329]
39.46268656716418
Finished Training for 80 % data.

85%|████████▌ | 17/20 [09:39<02:44, 54.83s/it]

[0.9337672591209412, 0.9243570566177368, 0.9225271344184875, 0.9209569692611694, 0.9190657734870911, 0.9190532565116882, 0.9180044531822205, 0.9180271029472351, 0.9176235198974609, 0.9163683652877808, 0.9156549572944641, 0.9148226380348206, 0.91883784532547, 0.9135919809341431, 0.913916826248169, 0.9141421914100647, 0.9
135724902153015, 0.9124138355255127, 0.9113626480102539, 0.9114129543304443]
39.46268656716418
Finished Training for 85 % data.

90%|█████████ | 18/20 [10:46<01:56, 58.35s/it]

[0.9335121512413025, 0.9243603348731995, 0.9229304194450378, 0.9210562109947205, 0.9187030792236328, 0.9188327193260193, 0.9180185198783875, 0.9165043234825134, 0.9160262942314148, 0.9160195589065552, 0.922141969203949, 0.9151712656021118, 0.9183416962623596, 0.9137123227119446, 0.9131589531898499, 0.9126260280609131, 0.9126508831977844, 0.9121503233909607, 0.91208416223526, 0.9132943749427795]
39.46268656716418
Finished Training for 90 % data.

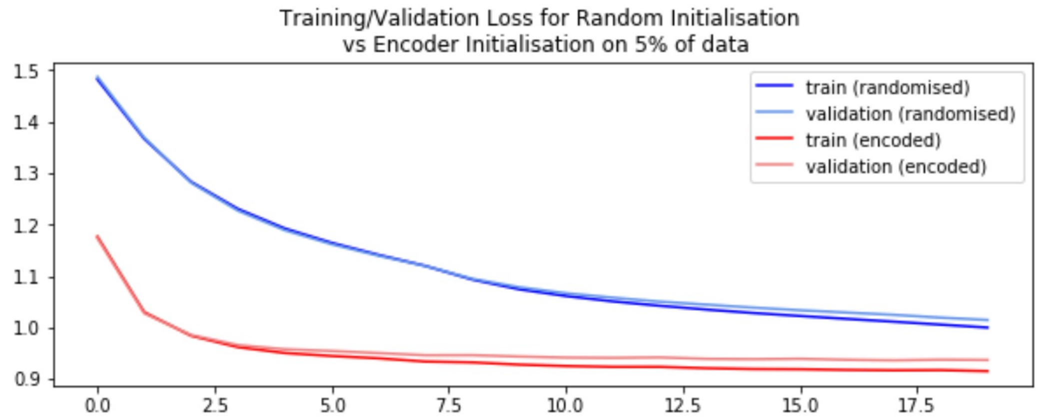
95%|█████████▌| 19/20 [11:50<01:00, 60.23s/it]

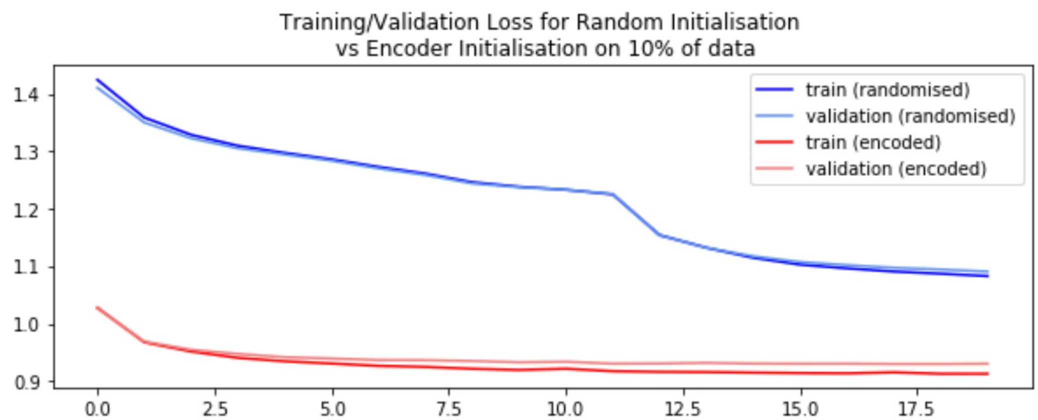
```
[0.9327353835105896, 0.9254993200302124, 0.9236725568771362, 0.9231318831443787,
0.9210036396980286, 0.9174844026565552, 0.9182441234588623, 0.9156839847564697,
0.9160642027854919, 0.9154162406921387, 0.9163666367530823, 0.9149297475814819,
0.9146223664283752, 0.9128500819206238, 0.913982629776001, 0.912638783454895, 0.
9123407602310181, 0.9140002727508545, 0.9112207889556885, 0.9124392867088318]
39.16417910447761
Finished Training for  95 % data.
```

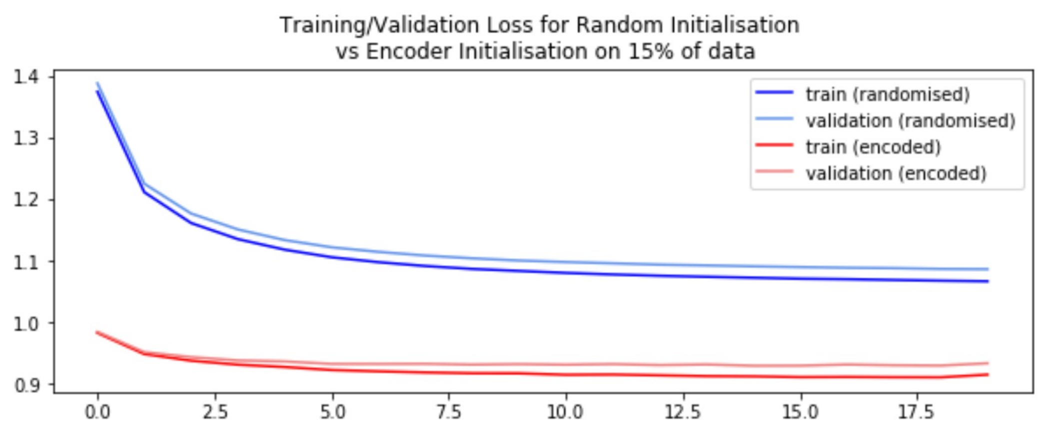
```
100%|██████████| 20/20 [12:58<00:00, 62.42s/it]
```

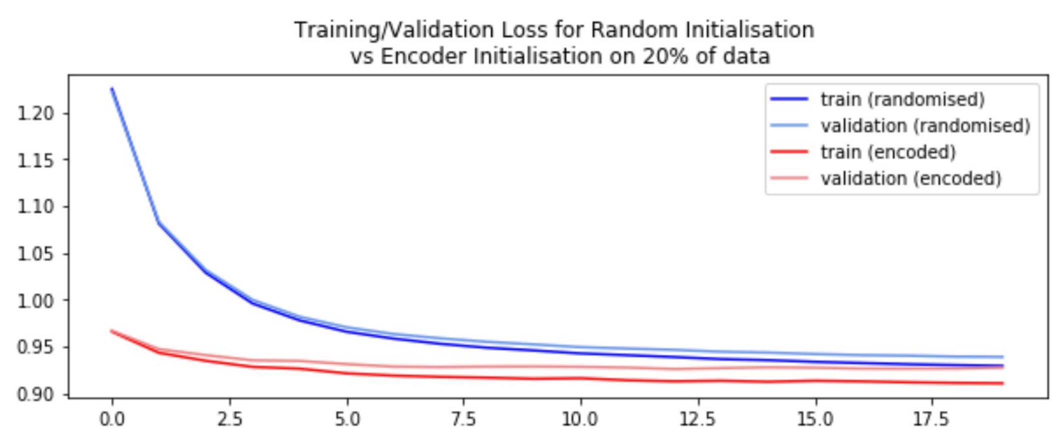
```
[0.9300309419631958, 0.9254710674285889, 0.9211783409118652, 0.9204171895980835,
0.9187513589859009, 0.9189250469207764, 0.9168115258216858, 0.9176651835441589,
0.9166386127471924, 0.9157572388648987, 0.9144414067268372, 0.914825975894928,
0.9148055911064148, 0.9133702516555786, 0.9140079021453857, 0.9127316474914551,
0.9122751951217651, 0.9130299091339111, 0.912039041519165, 0.9115883111953735]
39.54228855721393
Finished Training for  100 % data.
```

In [0]:
```
fig, axes = plt.subplots(20, 1, figsize=(10, 100))
fig.subplots_adjust(hspace=0.5)

for percent in random.keys():
  ax = axes[(percent//5 - 1)]
  train_loss_rand = random[percent]['train']
  valid_loss_rand = random[percent]['valid']
  train_loss_pretr = pretrained[percent]['train']
  valid_loss_pretr = pretrained[percent]['valid']
  ax.set_title('Training/Validation Loss for Random Initialisation \n vs Encoder In
itialisation on {}% of data'.format(percent))
  ax.plot(train_loss_rand, label='train (randomised)', c='b')
  ax.plot(valid_loss_rand, label='validation (randomised)', c='cornflowerblue')
  ax.plot(train_loss_pretr, label='train (encoded)', c='red')
  ax.plot(valid_loss_pretr, label='validation (encoded)', c='lightcoral')
  ax.legend()
```


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 5% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 10% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
1.4
1.3
1.2
1.1
1.0
0.9
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 15% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
1.4
1.3
1.2
1.1
1.0
0.9
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 20% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
1.20
1.15
1.10
1.05
1.00
0.95
0.90
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Comments

We can clearly see that pretraining the network is almost always superior. The pretrained network tends to begin with lower initial loss. This effect seems to be more pronounced the smaller the percentage of training data the model is given access to. The gradient of the loss also appears to be shallower for the pretrained network than for the random initialisation. These two factors perhaps agree with what we might expect: pretraining a model for a pretraining task 'familiarises' the model with the broad features of the underlying distribution, so that when the model is used for the primary task, a lot of the 'overhead' has already been dealt with. This would mean the model has lower initial loss, and is 'more converged' than the randomly initialised model. We might expect that if we trained these models fully to convergence, we would see that the models converge to the same loss, just that the pretrained model has a head start handling a similar distribution.

5. Provide the final accuracy on the training, validation, and test set for the best model you obtained for each of the two initialisation strategies

In [0]:
```
testys_1_position_tensor = labels_to_positions(testys_1_tensor, labels_1_to_positio
n)
```

In [0]:
```
epochs = 50
live=True

train_loss = []
valid_loss = []
test_loss = []
# best encoded initialisation model

# define model, loss and optimiser
mlp_pretrained = MLP()
# initialise model weights
mlp_pretrained.load_hiddens(autoencoder.encoder.state_dict())

if torch.cuda.is_available():
  mlp_pretrained.cuda()
loss = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(mlp_pretrained.parameters(), lr=0.0001,
                            weight_decay=1e-5)

# train model
for epoch in range(epochs):  # loop over the dataset multiple times
  for i, data in enumerate(train_1_loader, 0):
    # get the inputs; data is a list of [inputs, labels]
    inputs, labels = data
    labels = labels_to_positions(labels, labels_1_to_position)
    inputs = inputs

    if torch.cuda.is_available():
      inputs, labels = inputs.cuda(), labels.cuda()
    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = mlp_pretrained(inputs)
    loss_val = loss(outputs, labels)
    loss_val.backward()
    optimizer.step()

  if torch.cuda.is_available():
    train_loss.append(loss(mlp_pretrained(trainxs_perc.cuda()), trainys_perc.cuda
()).item())
    valid_loss.append(loss(mlp_pretrained(validxs_1_tensor.cuda()), validys_1_posit
ion_tensor.cuda()).item())
    test_loss.append(loss(mlp_pretrained(testxs_1_tensor.cuda()), testys_1_position
_tensor.cuda()).item())
  else:
    train_loss.append(loss(mlp_pretrained(trainxs_perc), trainys_perc).item())
    valid_loss.append(loss(mlp_pretrained(validxs_1_tensor), validys_1_position_ten
sor).item())
    test_loss.append(loss(mlp_pretrained(testxs_1_tensor), testys_1_position_tenso
r).item())


print('Best pretrained training loss: ', train_loss[-1])
print('Best pretrained validation loss: ', valid_loss[-1])
print('Best pretrained test loss: ', test_loss[-1])
print('Best pretrained training accuracy: ', accuracy(mlp_pretrained, trainxs_1_ten
sor.cuda(), trainys_1))
print('Best pretrained validation accuracy: ', accuracy(mlp_pretrained, validxs_1_t
ensor.cuda(), validys_1))
print('Best pretrained test accuracy:', accuracy(mlp_pretrained, testxs_1_tensor.cu
da(), testys_1))
```

```
Best pretrained training loss:  0.908551037311554
Best pretrained validation loss:  0.9200505018234253
Best pretrained test loss:  0.9201807379722595
Best pretrained training accuracy:  39.771771771771775
Best pretrained validation accuracy:  39.601990049751244
Best pretrained test accuracy: 39.08
```

In [0]:
```
epochs = 50
live=True

train_loss = []
valid_loss = []
test_loss = []
# best encoded initialisation model

# define model, loss and optimiser
mlp_random = MLP()

if torch.cuda.is_available():
  mlp_random.cuda()
loss = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(mlp_random.parameters(), lr=0.0001,
                            weight_decay=1e-5)

# train model
for epoch in range(epochs):  # loop over the dataset multiple times
  for i, data in enumerate(train_1_loader, 0):
    # get the inputs; data is a list of [inputs, labels]
    inputs, labels = data
    labels = labels_to_positions(labels, labels_1_to_position)
    inputs = inputs

    if torch.cuda.is_available():
      inputs, labels = inputs.cuda(), labels.cuda()
    # zero the parameter gradients
    optimizer.zero_grad()

    # forward + backward + optimize
    outputs = mlp_random(inputs)
    loss_val = loss(outputs, labels)
    loss_val.backward()
    optimizer.step()

  if torch.cuda.is_available():
    train_loss.append(loss(mlp_random(trainxs_perc.cuda()), trainys_perc.cuda()).it
em())
    valid_loss.append(loss(mlp_random(validxs_1_tensor.cuda()), validys_1_position_
tensor.cuda()).item())
    test_loss.append(loss(mlp_random(testxs_1_tensor.cuda()), testys_1_position_ten
sor.cuda()).item())
  else:
    train_loss.append(loss(mlp_random(trainxs_perc), trainys_perc).item())
    valid_loss.append(loss(mlp_random(validxs_1_tensor), validys_1_position_tenso
r).item())
    test_loss.append(loss(mlp_random(testxs_1_tensor), testys_1_position_tensor).it
em())


print('Best training random loss: ', train_loss[-1])
print('Best validation random loss: ', valid_loss[-1])
print('Best test random loss: ', test_loss[-1])
print('Best training random accuracy: ', accuracy(mlp_random, trainxs_1_tensor.cuda
(), trainys_1))
print('Best validation random accuracy: ', accuracy(mlp_random, validxs_1_tensor.cu
da(), validys_1))
print('Best test random accuracy:', accuracy(mlp_random, testxs_1_tensor.cuda(), te
stys_1))
```

```
Best training loss:  0.9088180661201477
Best validation loss:  0.9209878444671631
Best test loss:  0.9210487008094788
Best training accuracy:  39.77977977977978
Best validation accuracy:  39.30348258706468
Best test accuracy: 39.04
```

Results:

- Best training random loss: 0.9088180661201477
- Best validation random loss: 0.9209878444671631
- Best test random loss: 0.9210487008094788
- Best training random accuracy: 39.77977977977978
- Best validation random accuracy: 39.30348258706468
- Best test random accuracy: 39.04

- Best pretrained training loss: 0.908551037311554
- Best pretrained validation loss: 0.9200505018234253
- Best pretrained test loss: 0.9201807379722595
- Best pretrained training accuracy: 39.771771771771775
- Best pretrained validation accuracy: 39.601990049751244
- Best pretrained test accuracy: 39.08

We can see that there's little difference between the best randomly initialised model and the best model initialised via the encoder... this makes sense! The pretrained initialisation will typically do better, but we might get a really good random initialisation that puts us close to a good local minima in our parameter space.

Q5

In [0]:
```
import os
import gzip
import numpy as np
os.system("pip install python-mnist")
from mnist import MNIST
import matplotlib.pyplot as plt
%matplotlib inline
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.autograd import Variable
from sklearn.metrics import confusion_matrix
import pandas as pd
from tqdm import tqdm
```

Setup

Download and process data

In [0]:
```
# Download the dataset.
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz -o /tmp/train-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz -o /tmp/train-labels-idx1-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz -o /tmp/t10k-images-idx3-ubyte.gz')
os.system('curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz -o /tmp/t10k-labels-idx1-ubyte.gz')
pass
```

In [0]:
```
# load the dataset with shape (*, 784)
mnistdata = MNIST("/tmp")
mnistdata.gz = True
trainxs_raw, trainys_raw = mnistdata.load_training()
testxs_raw, testys_raw = mnistdata.load_testing()

# reshape data into square image (*, 1, 28, 28)
trainxs_square  = np.array(trainxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
trainys_square  = np.array(trainys_raw, dtype=np.long)
testxs          = np.array(testxs_raw, dtype=np.float32).reshape(-1, 1, 28, 28)
testys          = np.array(testys_raw, dtype=np.long)

# split training dataset
trainxs = trainxs_square[:50000]
trainys = trainys_square[:50000]
validxs = trainxs_square[50000:]
validys = trainys_square[50000:]
```

In [0]:
```
# process into FashionMNist 1 and Fashon MNist 2

def filter_for_labels(trainxs, trainys, validxs, validys, testxs, testys, labels):
    # FashinoMNist 1
    train_ix = np.isin(trainys, labels)
    trainxs_ = trainxs[train_ix]
    trainys_ = trainys[train_ix]

    val_ix = np.isin(validys, labels)
    validxs_ = validxs[val_ix]
    validys_ = validys[val_ix]

    test_ix = np.isin(testys, labels)
    testxs_ = testxs[test_ix]
    testys_ = testys[test_ix]

    return trainxs_, trainys_, validxs_, validys_, testxs_, testys_

labels_1 = [0, 1, 4, 5, 8]
labels_2 = [x for x in list(range(10)) if x not in labels_1]

# FashinoMNist1
trainxs_1, trainys_1, validxs_1, validys_1, testxs_1, testys_1 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_1
)

def change_labels(ys, labels):
  new_ys = np.zeros(len(ys))
  position_dict = {labels[i]: i for i in range(len(labels))}
  for label in position_dict.keys():
    new_ys[ys == label] = position_dict[label]
  return new_ys

#def labels_to_positions(labels, labels_to_positions):
#  positions = np.full(len(labels), -1)
#  for label, position in labels_to_positions.items():
#    ix = (labels == label).nonzero()
#    positions[ix] = position
#  if (positions < 0).any():
#    print('WARNING! invalid label index!')
#  return torch.from_numpy(positions)

trainys_1 = change_labels(trainys_1, labels_1)
validys_1 = change_labels(validys_1, labels_1)
testys_1 = change_labels(testys_1, labels_1)

trainys_1 = np.array(trainys_1, dtype=np.long)
validys_1 = np.array(validys_1, dtype=np.long)
testys_1 = np.array(testys_1, dtype=np.long)

#FashionMNist2
trainxs_2, trainys_2, validxs_2, validys_2, testxs_2, testys_2 = filter_for_labels(
    trainxs, trainys, validxs, validys, testxs, testys, labels_2
)

trainys_2 = change_labels(trainys_2, labels_2)
validys_2 = change_labels(validys_2, labels_2)
testys_2 = change_labels(testys_2, labels_2)

trainys_2 = np.array(trainys_2, dtype=np.long)
validys_2 = np.array(validys_2, dtype=np.long)
testys_2 = np.array(testys_2, dtype=np.long)
```

Structure dataset

In [0]:
```
class DatasetConstructor(Dataset):

    def __init__(self, X, Y):
        self.X = torch.from_numpy(X)
        self.Y = torch.from_numpy(Y)
        self.len = X.shape[0]

    def __getitem__(self, index):
        return self.X[index], self.Y[index]

    def __len__(self):
        return self.len

BATCHSIZE = 16

train_data_1 = DatasetConstructor(trainxs_1, trainys_1)
valid_data_1 = DatasetConstructor(validxs_1, validys_1)
test_data_1  = DatasetConstructor(testxs_1, testys_1)

train_data_2 = DatasetConstructor(trainxs_2, trainys_2)
valid_data_2 = DatasetConstructor(validxs_2, validys_2)
test_data_2 = DatasetConstructor(testxs_2, testys_2)

train_loader_1  = DataLoader(dataset=train_data_1, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
valid_loader_1  = DataLoader(dataset=valid_data_1, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
test_loader_1   = DataLoader(dataset=test_data_1, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)

train_loader_2  = DataLoader(dataset=train_data_2, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
valid_loader_2  = DataLoader(dataset=valid_data_2, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)
test_loader_2   = DataLoader(dataset=test_data_2, batch_size=BATCHSIZE,
                          shuffle = True, num_workers=2)

# tensor dataset for loss and accuracy computation
trainxs_tensor_1 = torch.from_numpy(trainxs_1)
trainys_tensor_1 = torch.from_numpy(trainys_1)
validxs_tensor_1 = torch.from_numpy(validxs_1)
validys_tensor_1 = torch.from_numpy(validys_1)
testxs_tensor_1  = torch.from_numpy(testxs_1)
testys_tensor_1  = torch.from_numpy(testys_1)

trainxs_tensor_2 = torch.from_numpy(trainxs_2)
trainys_tensor_2 = torch.from_numpy(trainys_2)
validxs_tensor_2 = torch.from_numpy(validxs_2)
validys_tensor_2 = torch.from_numpy(validys_2)
testxs_tensor_2  = torch.from_numpy(testxs_2)
testys_tensor_2  = torch.from_numpy(testys_2)
```

1. Implement a multi-class, convolutional neural network with cross-entropy loss for the Fashion-MNIST-2 data

In [0]:
```
# Class to define model of arbitrarily sized architecture

class CNN_class(nn.Module):

    def __init__(self, input_size, channels_conv, pool_size, kernel_size, hids, out
put_size):
        super(CNN_class, self).__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.channels_conv = channels_conv
        self.kernel_size = kernel_size
        self.pool_size = pool_size
        self.hids = hids
        self.conv_layers = []
        self.classification_layers = []

        assert len(channels_conv) > 0
        assert len(hids) > 0
        assert len(channels_conv) == len(kernel_size) == len(pool_size)
        assert all(i > 0 for i in hids)
        assert all(i > 0 for i in channels_conv)
        assert all(i > 0 for i in kernel_size)

        self.conv_layers.append(nn.Conv2d(1, self.channels_conv[0], self.kernel_siz
e[0], padding=1, padding_mode="same"))
        self.conv_layers.append(nn.BatchNorm2d(self.channels_conv[0]))
        self.conv_layers.append(nn.ReLU(inplace=True))
        self.conv_layers.append(nn.MaxPool2d(self.pool_size[0],self.pool_size[0]))

        for i in range(1,len(self.channels_conv)):
          self.conv_layers.append(nn.Conv2d(self.channels_conv[i-1], self.channels_
conv[i], self.kernel_size[i], padding=1, padding_mode="same"))
          self.conv_layers.append(nn.BatchNorm2d(self.channels_conv[i]))
          self.conv_layers.append(nn.ReLU(inplace=True))
          self.conv_layers.append(nn.MaxPool2d(self.pool_size[i],self.pool_size
[i]))

        self.conv = nn.Sequential(*self.conv_layers)

        p = np.array(self.input_size) // self.pool_size[0][0]
        for i in range(1,len(self.channels_conv)):
          p = p // self.pool_size[i][0]

        self.conv_out_size = int(p[0]*p[1]*self.channels_conv[-1])

        self.classification_layers.append(nn.Linear(self.conv_out_size, self.hids
[0]))
        self.classification_layers.append(nn.ReLU(inplace=True))

        for i in range(1,len(hids)):
          self.classification_layers.append(nn.Linear(self.hids[i-1], self.hids
[i]))

          self.classification_layers.append(nn.ReLU(inplace=True))

        self.classification_layers.append(nn.Linear(self.hids[-1], self.output_siz
e))

        self.classif = nn.Sequential(*self.classification_layers)

    def forward(self, x):

      x = self.conv(x)
      x = x.view(x.size(0), -1)      # reshape
      x = self.classif(x)
```

In [0]:
```
def accuracy(model, X, Y):
    outputs = model(X)
    _, prediction = torch.max(outputs, 1)
    if torch.cuda.is_available():
        prediction = prediction.cpu()
    return sum((Y - prediction.numpy())==0) / len(Y) * 100
```

In [0]:
```
# Training function for input training parameters

def train_CNN(model, lr, reg, lambd, num_of_epochs, opt='', mnist=1):

  # Generic function to train model with input training parameters:
  # lr = learning rate
  # reg = 1 or 2 for L1 or L2 regularisation
  # lambd = parameter lambda of regularisation - set to lambda = 0 for no regularis
ation
  # num_of_epochs = number of epochs for training loop
  # opt = keyword argument set to SGD or ADAM

  if mnist==1:
    train_loader = train_loader_1
    trainxs_tensor = trainxs_tensor_1
    trainys_tensor = trainys_tensor_1
    validxs_tensor = validxs_tensor_1
    validys_tensor = validys_tensor_1
    testxs_tensor  = testxs_tensor_1
    testys_tensor  = testys_tensor_1
    trainys = trainys_1
    validys = validys_1
    testys = testys_1

  if mnist==2:
    train_loader = train_loader_2
    trainxs_tensor = trainxs_tensor_2
    trainys_tensor = trainys_tensor_2
    validxs_tensor = validxs_tensor_2
    validys_tensor = validys_tensor_2
    testxs_tensor  = testxs_tensor_2
    testys_tensor  = testys_tensor_2
    trainys = trainys_2
    validys = validys_2
    testys = testys_2

  if torch.cuda.is_available():
    model.cuda()

  loss = nn.CrossEntropyLoss()
  if opt == 'SGD':
    optimizer = optim.SGD(model.parameters(), lr=lr)
  else:
    optimizer = optim.Adam(model.parameters(), lr=lr)

  train_loss = []
  valid_loss = []
  test_loss = []

  for epoch in tqdm(range(num_of_epochs)):

    running_loss = 0.0
    running_loss_without_reg = 0.0
    for i, data in enumerate(train_loader, 0):
      inputs, labels = data

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = model(inputs)
```

2. Iteratively tune your model structure and hyperparameters using the validation set of Fashion-MNIST-2, until you arrive at a model performance you are comfortable with.

In [0]:
```
# Create an initial data frame
data_frame = {'Convolution layers':[],
        'Classification hidden units':[],
        'Optimiser':[], 'Learning rate':[], 'Regularisation/ parameter (lambd
a)':[],
        'Train_Loss':[], 'Valid_Loss':[], 'Test_Loss':[],
        'Train_accuracy (%)':[], 'Valid_accuracy (%)':[], 'Test_accuracy (%)':[]
       }
df = pd.DataFrame(data_frame)
```

In [0]:
```
# Function to train and append results to a data frame
def train_and_append(df, input_size, channels_conv, pool_size, kernel_size, hids, o
utput_size, lr, reg, lmbd, num_of_epochs, opt, mnist=1):


  model = CNN_class(input_size, channels_conv, pool_size, kernel_size, hids, output
_size)
  train_loss, valid_loss, test_loss, final_accuracies = train_CNN(model, lr, reg, l
mbd, num_of_epochs, opt=opt, mnist=mnist)

  df = df.append({'Train_accuracy (%)': final_accuracies[0], 'Valid_accuracy (%)':
final_accuracies[1], 'Test_accuracy (%)': final_accuracies[2],
                'Train_Loss': train_loss[-1], 'Valid_Loss': valid_loss[-1], 'Test_L
oss': test_loss[-1],
                'Convolution layers': channels_conv, 'Classification hidden units':
hids, 'Optimiser': str(opt), 'Learning rate': lr,
                'Regularisation/ parameter (lambda)': "L {}, lambda = {}".format(re
g, lmbd)}, ignore_index=True)

  return df
```

In [0]:

```python
# Append results to the data frame above

df = train_and_append(df, [28,28], [4,4], [[2,2],[2,2]], [3,3], [32], 5, 0.001, 1,
0.001, 20, 'SGD', mnist=2)  # model 2
df = train_and_append(df, [28,28], [10,10,10], [[2,2],[2,2],[2,2]], [3,3,3], [64],
5, 0.001, 2,  0.001, 20, 'SGD', mnist=2) # model 3
df = train_and_append(df, [28,28], [10,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.001, 1, 0.001, 20, 'SGD', mnist=2) # model 4
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [64,3
2], 5, 0.001, 2, 0.01, 20, 'SGD', mnist=2) # model 5
```

```
100%|██████████| 20/20 [02:12<00:00,  6.62s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [02:51<00:00,  8.61s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [02:39<00:00,  7.92s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [03:05<00:00,  9.43s/it]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>
```

In [0]:

```
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.001, 2, 0.01, 20, 'SGD', mnist=2) # model 6
df = train_and_append(df, [28,28], [10,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [64,6
4], 5, 0.001, 2, 0.001, 20, 'ADAM', mnist=2) # model 7
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128,1
28], 5, 0.0001, 2, 0.01, 20, 'ADAM', mnist=2) # model 8
df = train_and_append(df, [28,28], [15,15,15], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.0001, 2, 0.01, 20, 'ADAM', mnist=2) #model 9
df = train_and_append(df, [28,28], [20,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [128],
5, 0.0001, 2, 0.01, 20, 'ADAM',mnist=2) # model 10
```

```
100%|██████████| 20/20 [02:53<00:00,  8.62s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [04:07<00:00, 12.30s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [04:11<00:00, 12.52s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [03:49<00:00, 11.49s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>

100%|██████████| 20/20 [03:51<00:00, 11.43s/it]

<class 'torch.Tensor'>
<class 'torch.Tensor'>
<class 'torch.Tensor'>
```

In [0]:
```
display(df)
```

	Convolution layers	Classification hidden units	Optimiser	Learning rate	Regularisation/ parameter (lambda)	Train_Loss	Valid_Loss	Test_Loss	Train_a
0	[4, 4]	[32]	SGD	0.0010	L 1, lambda = 0.001	0.226389	0.233475	0.256004	9
1	[10, 10, 10]	[64]	SGD	0.0010	L 2, lambda = 0.001	0.184370	0.199692	0.232483	9
2	[10, 15, 15]	[128]	SGD	0.0010	L 1, lambda = 0.001	0.201961	0.219638	0.240082	9
3	[15, 15, 15]	[64, 32]	SGD	0.0010	L 2, lambda = 0.01	0.155859	0.185192	0.206011	9
4	[15, 15, 15]	[128]	SGD	0.0010	L 2, lambda = 0.01	0.148704	0.175516	0.198037	9
5	[10, 20, 20]	[64, 64]	ADAM	0.0010	L 2, lambda = 0.001	0.055808	0.213193	0.242133	9
6	[15, 15, 15]	[128, 128]	ADAM	0.0001	L 2, lambda = 0.01	0.100377	0.160425	0.183457	9
7	[15, 15, 15]	[128]	ADAM	0.0001	L 2, lambda = 0.01	0.100517	0.169136	0.196757	9
8	[20, 20, 20]	[128]	ADAM	0.0001	L 2, lambda = 0.01	0.075787	0.170292	0.200024	9

After testing we decided to implement model 8 as it performed the best on the validation set

In [0]:
```
mod_best =  CNN_class([28,28], [20,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [128], 5)
train_loss, valid_loss, test_loss, final_accuracies = train_CNN(mod_best, 0.0001,
2, 0.01, 20, opt='ADAM', mnist=2)
torch.save(mod_best.state_dict(), 'drive/My Drive/DL_Models/model_fin.pth')
```

```
100%|██████████| 20/20 [03:52<00:00, 11.63s/it]
```

1. Implement a multi-class, convolutional neural network with cross-entropy loss for the Fashion-MNIST-1 data, which shares the same structure as the one you used for the Fashion-MNIST-2 data.

&

1. Compare using random weights to those obtained by training on Fashion-MNIST-2 - you should randomly re-initialise the classification layer though - to initialise the multi-class, convolutional neural network by plotting the training and validation loss for both options when you use 5%, 10%, ..., 100% of the available Fashion-MNIST-1 training data to train your model. Is there a point where one initialisation option is superior to the other? Is one option always superior to the other?

In [0]:
```
# Function to randomly re-initialise the classification layer
from random import random
def init_weights(m):
    if type(m) == nn.Linear:
        torch.nn.init.xavier_uniform_(m.weight)
        m.bias.data.uniform_()
```

In [0]:
```
import pandas as pd
percentages = list(range(5, 105, 5))
random = {percentage: {'valid': [], 'train': []} for percentage in percentages}
pretrained = {percentage: {'valid': [], 'train': []} for percentage in percentages}

labels_1 = [0, 1, 4, 5, 8]
labels_2 = [x for x in list(range(10)) if x not in labels_1]
labels_1_to_position = {labels_1[i]: i for i in range(len(labels_1))}
labels_2_to_position = {labels_2[i]: i for i in range(len(labels_2))}
position_to_labels_1 = {v: k for k, v in labels_1_to_position.items()}
position_to_labels_2 = {v: k for k, v in labels_2_to_position.items()}
```

In [0]:
```
def build_dataset(xs, ys, percentage=100):
  m = xs.shape[0]
  assert(m == ys.shape[0])
  cutoff = int((percentage/100)*m)
  permutation_ix = np.random.permutation(list(range(m)))
  xs_perm = xs[permutation_ix]
  ys_perm = ys[permutation_ix]
  return DatasetConstructor(xs_perm[:cutoff], ys_perm[:cutoff])

def build_dataloader(xs, ys, percentage=100, batch_size=BATCHSIZE):
  dataset = build_dataset(xs, ys, percentage)
  return DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=2)
```

Training using randomly initialised weights.

In [0]:
```
def accuracy_q5(model, X, Y):
    outputs = model(X)
    _, prediction = torch.max(outputs, 1)
    if torch.cuda.is_available():
        prediction = prediction.cpu()
        Y = Y.cpu()
    return sum((Y.numpy() - prediction.numpy())==0) / len(Y) * 100
```

In [0]:
```
from tqdm import tqdm
random_res = {percentage: {'valid_loss': [], 'train_loss': [], 'valid_acc': [], 'tr
ain_acc': [],'test_acc': []} for percentage in percentages}
epochs = 20
live=False

# random initialisations
for percent in tqdm(random_res.keys()):

  # build training data
  train_perc_loader = build_dataloader(trainxs_1, trainys_1, percentage=percent)
  trainxs_perc = train_perc_loader.dataset.X
  trainys_perc = train_perc_loader.dataset.Y
  validxs_tensor = validxs_tensor_1
  validys_tensor = validys_tensor_1
  testxs_tensor = testxs_tensor_1
  testys_tensor = testys_tensor_1

  #build model

  # define model, loss and optimiser
  cnn = CNN_class([28,28], [20,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [128], 5)

  if torch.cuda.is_available():
    cnn.cuda()

  loss = nn.CrossEntropyLoss()
  optimizer = optim.Adam(cnn.parameters(), lr=0.0001)

  train_loss = []
  valid_loss = []
  test_loss = []

  for epoch in tqdm(range(epochs)):

    running_loss = 0.0
    running_loss_without_reg = 0.0
    for i, data in enumerate(train_perc_loader, 0):
      inputs, labels = data

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = cnn(inputs)
      loss_val_pre = loss(outputs, labels)
      regularisation = 0
      for param in cnn.parameters():
          regularisation += torch.norm(param, 2)
      loss_val = loss_val_pre + 0.01*regularisation
      loss_val.backward()
      optimizer.step()

    if torch.cuda.is_available():
      random_res[percent]['train_loss'].append(loss(cnn(trainxs_perc.cuda()), train
ys_perc.cuda()).item())
      random_res[percent]['valid_loss'].append(loss(cnn(validxs_tensor.cuda()), val
idys_tensor.cuda()).item())
      random_res[percent]['train_acc'].append(accuracy_q5(cnn, trainxs_perc.cuda(),
trainys_perc.cuda()))
      random_res[percent]['valid_acc'].append(accuracy_q5(cnn, validxs_tensor.cuda
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:00<00:15,  1.21it/s]
 10%|█         | 2/20 [00:01<00:14,  1.22it/s]
 15%|█▌        | 3/20 [00:02<00:13,  1.22it/s]
 20%|██        | 4/20 [00:03<00:13,  1.23it/s]
 25%|██▌       | 5/20 [00:04<00:12,  1.22it/s]
 30%|███       | 6/20 [00:04<00:11,  1.24it/s]
 35%|███▌      | 7/20 [00:05<00:10,  1.24it/s]
 40%|████      | 8/20 [00:06<00:09,  1.23it/s]
 45%|████▌     | 9/20 [00:07<00:08,  1.23it/s]
 50%|█████     | 10/20 [00:08<00:08,  1.24it/s]
 55%|█████▌    | 11/20 [00:08<00:07,  1.24it/s]
 60%|██████    | 12/20 [00:09<00:06,  1.24it/s]
 65%|██████▌   | 13/20 [00:10<00:05,  1.24it/s]
 70%|███████   | 14/20 [00:11<00:04,  1.25it/s]
 75%|███████▌  | 15/20 [00:12<00:04,  1.25it/s]
 80%|████████  | 16/20 [00:12<00:03,  1.24it/s]
 85%|████████▌ | 17/20 [00:13<00:02,  1.24it/s]
 90%|█████████ | 18/20 [00:14<00:01,  1.23it/s]
 95%|█████████▌| 19/20 [00:15<00:00,  1.22it/s]
100%|██████████| 20/20 [00:16<00:00,  1.23it/s]
  5%|▌         | 1/20 [00:16<05:08, 16.24s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  5 % data.

  5%|▌         | 1/20 [00:01<00:26,  1.38s/it]
 10%|█         | 2/20 [00:02<00:25,  1.40s/it]
 15%|█▌        | 3/20 [00:04<00:23,  1.39s/it]
 20%|██        | 4/20 [00:05<00:22,  1.38s/it]
 25%|██▌       | 5/20 [00:06<00:20,  1.38s/it]
 30%|███       | 6/20 [00:08<00:19,  1.39s/it]
 35%|███▌      | 7/20 [00:09<00:18,  1.39s/it]
 40%|████      | 8/20 [00:11<00:16,  1.38s/it]
 45%|████▌     | 9/20 [00:12<00:15,  1.39s/it]
 50%|█████     | 10/20 [00:13<00:13,  1.39s/it]
 55%|█████▌    | 11/20 [00:15<00:12,  1.39s/it]
 60%|██████    | 12/20 [00:16<00:11,  1.39s/it]
 65%|██████▌   | 13/20 [00:18<00:09,  1.39s/it]
 70%|███████   | 14/20 [00:19<00:08,  1.39s/it]
 75%|███████▌  | 15/20 [00:20<00:06,  1.38s/it]
 80%|████████  | 16/20 [00:22<00:05,  1.38s/it]
 85%|████████▌ | 17/20 [00:23<00:04,  1.37s/it]
 90%|█████████ | 18/20 [00:24<00:02,  1.37s/it]
 95%|█████████▌| 19/20 [00:26<00:01,  1.37s/it]
100%|██████████| 20/20 [00:27<00:00,  1.45s/it]
 10%|█         | 2/20 [00:44<05:55, 19.76s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  10 % data.
```

```
  5%|▌         | 1/20 [00:02<00:38,  2.01s/it]
 10%|█         | 2/20 [00:03<00:35,  1.98s/it]
 15%|█▌        | 3/20 [00:05<00:33,  2.00s/it]
 20%|██        | 4/20 [00:08<00:32,  2.05s/it]
 25%|██▌       | 5/20 [00:10<00:30,  2.05s/it]
 30%|███       | 6/20 [00:12<00:28,  2.03s/it]
 35%|███▌      | 7/20 [00:14<00:25,  1.99s/it]
 40%|████      | 8/20 [00:16<00:23,  1.98s/it]
 45%|████▌     | 9/20 [00:18<00:21,  1.98s/it]
 50%|█████     | 10/20 [00:19<00:19,  1.98s/it]
 55%|█████▌    | 11/20 [00:21<00:17,  1.99s/it]
 60%|██████    | 12/20 [00:23<00:15,  1.98s/it]
 65%|██████▌   | 13/20 [00:25<00:13,  1.97s/it]
 70%|███████   | 14/20 [00:27<00:11,  1.96s/it]
 75%|███████▌  | 15/20 [00:29<00:09,  1.97s/it]
 80%|████████  | 16/20 [00:31<00:07,  1.95s/it]
 85%|████████▌ | 17/20 [00:33<00:05,  1.94s/it]
 90%|█████████ | 18/20 [00:35<00:03,  1.95s/it]
 95%|█████████▌| 19/20 [00:37<00:01,  1.95s/it]
100%|██████████| 20/20 [00:39<00:00,  1.95s/it]
 15%|█▌        | 3/20 [01:23<07:16, 25.70s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  15 % data.

  5%|▌         | 1/20 [00:02<00:48,  2.57s/it]
 10%|█         | 2/20 [00:05<00:46,  2.57s/it]
 15%|█▌        | 3/20 [00:07<00:43,  2.56s/it]
 20%|██        | 4/20 [00:10<00:40,  2.54s/it]
 25%|██▌       | 5/20 [00:12<00:38,  2.54s/it]
 30%|███       | 6/20 [00:15<00:35,  2.54s/it]
 35%|███▌      | 7/20 [00:17<00:32,  2.54s/it]
 40%|████      | 8/20 [00:20<00:31,  2.63s/it]
 45%|████▌     | 9/20 [00:23<00:28,  2.63s/it]
 50%|█████     | 10/20 [00:25<00:26,  2.62s/it]
 55%|█████▌    | 11/20 [00:28<00:24,  2.67s/it]
 60%|██████    | 12/20 [00:31<00:21,  2.63s/it]
 65%|██████▌   | 13/20 [00:33<00:18,  2.59s/it]
 70%|███████   | 14/20 [00:36<00:15,  2.58s/it]
 75%|███████▌  | 15/20 [00:38<00:12,  2.58s/it]
 80%|████████  | 16/20 [00:41<00:10,  2.56s/it]
 85%|████████▌ | 17/20 [00:43<00:07,  2.55s/it]
 90%|█████████ | 18/20 [00:46<00:05,  2.53s/it]
 95%|█████████▌| 19/20 [00:48<00:02,  2.54s/it]
100%|██████████| 20/20 [00:51<00:00,  2.54s/it]
 20%|██        | 4/20 [02:15<08:55, 33.44s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  20 % data.
```

```
  5%|▌         | 1/20 [00:03<00:59,  3.14s/it]
 10%|█         | 2/20 [00:06<00:56,  3.14s/it]
 15%|█▌        | 3/20 [00:09<00:53,  3.13s/it]
 20%|██        | 4/20 [00:12<00:50,  3.13s/it]
 25%|██▌       | 5/20 [00:15<00:47,  3.16s/it]
 30%|███       | 6/20 [00:18<00:44,  3.15s/it]
 35%|███▌      | 7/20 [00:21<00:40,  3.14s/it]
 40%|████      | 8/20 [00:25<00:37,  3.11s/it]
 45%|████▌     | 9/20 [00:28<00:34,  3.12s/it]
 50%|█████     | 10/20 [00:31<00:31,  3.16s/it]
 55%|█████▌    | 11/20 [00:34<00:28,  3.15s/it]
 60%|██████    | 12/20 [00:37<00:25,  3.23s/it]
 65%|██████▌   | 13/20 [00:41<00:22,  3.20s/it]
 70%|███████   | 14/20 [00:44<00:19,  3.17s/it]
 75%|███████▌  | 15/20 [00:47<00:15,  3.16s/it]
 80%|████████  | 16/20 [00:50<00:12,  3.14s/it]
 85%|████████▌ | 17/20 [00:53<00:09,  3.13s/it]
 90%|█████████ | 18/20 [00:56<00:06,  3.12s/it]
 95%|█████████▌| 19/20 [00:59<00:03,  3.12s/it]
100%|██████████| 20/20 [01:02<00:00,  3.11s/it]
 25%|██▌       | 5/20 [03:18<10:34, 42.27s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  25 % data.

  5%|▌         | 1/20 [00:03<01:10,  3.69s/it]
 10%|█         | 2/20 [00:07<01:06,  3.70s/it]
 15%|█▌        | 3/20 [00:11<01:03,  3.71s/it]
 20%|██        | 4/20 [00:15<01:00,  3.77s/it]
 25%|██▌       | 5/20 [00:18<00:57,  3.80s/it]
 30%|███       | 6/20 [00:22<00:53,  3.80s/it]
 35%|███▌      | 7/20 [00:26<00:50,  3.87s/it]
 40%|████      | 8/20 [00:30<00:45,  3.82s/it]
 45%|████▌     | 9/20 [00:34<00:42,  3.87s/it]
 50%|█████     | 10/20 [00:38<00:38,  3.84s/it]
 55%|█████▌    | 11/20 [00:41<00:34,  3.79s/it]
 60%|██████    | 12/20 [00:45<00:29,  3.75s/it]
 65%|██████▌   | 13/20 [00:49<00:26,  3.74s/it]
 70%|███████   | 14/20 [00:52<00:22,  3.71s/it]
 75%|███████▌  | 15/20 [00:56<00:18,  3.71s/it]
 80%|████████  | 16/20 [01:00<00:14,  3.70s/it]
 85%|████████▌ | 17/20 [01:04<00:11,  3.70s/it]
 90%|█████████ | 18/20 [01:07<00:07,  3.70s/it]
 95%|█████████▌| 19/20 [01:11<00:03,  3.70s/it]
100%|██████████| 20/20 [01:15<00:00,  3.71s/it]
 30%|███       | 6/20 [04:33<12:09, 52.14s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  30 % data.
```

```
  5%|▌         | 1/20 [00:04<01:21,  4.31s/it]
 10%|█         | 2/20 [00:08<01:17,  4.30s/it]
 15%|█▌        | 3/20 [00:13<01:14,  4.38s/it]
 20%|██        | 4/20 [00:17<01:09,  4.34s/it]
 25%|██▌       | 5/20 [00:21<01:06,  4.41s/it]
 30%|███       | 6/20 [00:26<01:01,  4.37s/it]
 35%|███▌      | 7/20 [00:30<00:56,  4.34s/it]
 40%|████      | 8/20 [00:34<00:51,  4.32s/it]
 45%|████▌     | 9/20 [00:39<00:47,  4.32s/it]
 50%|█████     | 10/20 [00:43<00:43,  4.32s/it]
 55%|█████▌    | 11/20 [00:47<00:38,  4.31s/it]
 60%|██████    | 12/20 [00:52<00:34,  4.30s/it]
 65%|██████▌   | 13/20 [00:56<00:29,  4.28s/it]
 70%|███████   | 14/20 [01:00<00:25,  4.29s/it]
 75%|███████▌  | 15/20 [01:04<00:21,  4.28s/it]
 80%|████████  | 16/20 [01:09<00:17,  4.27s/it]
 85%|████████▌ | 17/20 [01:13<00:13,  4.36s/it]
 90%|█████████ | 18/20 [01:17<00:08,  4.33s/it]
 95%|█████████▌| 19/20 [01:22<00:04,  4.41s/it]
100%|██████████| 20/20 [01:26<00:00,  4.36s/it]
 35%|███▌      | 7/20 [06:00<13:32, 62.52s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  35 % data.

  5%|▌         | 1/20 [00:04<01:32,  4.85s/it]
 10%|█         | 2/20 [00:09<01:26,  4.83s/it]
 15%|█▌        | 3/20 [00:14<01:21,  4.81s/it]
 20%|██        | 4/20 [00:19<01:16,  4.81s/it]
 25%|██▌       | 5/20 [00:24<01:12,  4.81s/it]
 30%|███       | 6/20 [00:28<01:07,  4.85s/it]
 35%|███▌      | 7/20 [00:33<01:03,  4.85s/it]
 40%|████      | 8/20 [00:38<00:58,  4.85s/it]
 45%|████▌     | 9/20 [00:43<00:53,  4.85s/it]
 50%|█████     | 10/20 [00:48<00:48,  4.88s/it]
 55%|█████▌    | 11/20 [00:53<00:45,  5.01s/it]
 60%|██████    | 12/20 [00:58<00:39,  4.96s/it]
 65%|██████▌   | 13/20 [01:03<00:34,  4.91s/it]
 70%|███████   | 14/20 [01:08<00:29,  4.90s/it]
 75%|███████▌  | 15/20 [01:13<00:24,  4.86s/it]
 80%|████████  | 16/20 [01:17<00:19,  4.83s/it]
 85%|████████▌ | 17/20 [01:22<00:14,  4.87s/it]
 90%|█████████ | 18/20 [01:27<00:09,  4.88s/it]
 95%|█████████▌| 19/20 [01:32<00:04,  4.85s/it]
100%|██████████| 20/20 [01:37<00:00,  4.83s/it]
 40%|████      | 8/20 [07:37<14:35, 72.96s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  40 % data.
```

```
  5%|▌         | 1/20 [00:05<01:40,  5.30s/it]
 10%|█         | 2/20 [00:11<01:37,  5.43s/it]
 15%|█▌        | 3/20 [00:16<01:33,  5.52s/it]
 20%|██        | 4/20 [00:22<01:27,  5.50s/it]
 25%|██▌       | 5/20 [00:27<01:21,  5.46s/it]
 30%|███       | 6/20 [00:32<01:15,  5.41s/it]
 35%|███▌      | 7/20 [00:38<01:09,  5.37s/it]
 40%|████      | 8/20 [00:43<01:04,  5.39s/it]
 45%|████▌     | 9/20 [00:49<00:59,  5.40s/it]
 50%|█████     | 10/20 [00:54<00:53,  5.39s/it]
 55%|█████▌    | 11/20 [00:59<00:48,  5.41s/it]
 60%|██████    | 12/20 [01:05<00:43,  5.43s/it]
 65%|██████▌   | 13/20 [01:11<00:39,  5.62s/it]
 70%|███████   | 14/20 [01:17<00:33,  5.65s/it]
 75%|███████▌  | 15/20 [01:22<00:27,  5.58s/it]
 80%|████████  | 16/20 [01:27<00:22,  5.53s/it]
 85%|████████▌ | 17/20 [01:33<00:16,  5.52s/it]
 90%|█████████ | 18/20 [01:38<00:10,  5.49s/it]
 95%|█████████▌| 19/20 [01:44<00:05,  5.46s/it]
100%|██████████| 20/20 [01:49<00:00,  5.46s/it]
 45%|████▌     | 9/20 [09:27<15:23, 83.99s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  45 % data.

  5%|▌         | 1/20 [00:06<01:55,  6.05s/it]
 10%|█         | 2/20 [00:12<01:48,  6.04s/it]
 15%|█▌        | 3/20 [00:18<01:43,  6.08s/it]
 20%|██        | 4/20 [00:24<01:36,  6.05s/it]
 25%|██▌       | 5/20 [00:30<01:31,  6.10s/it]
 30%|███       | 6/20 [00:36<01:25,  6.08s/it]
 35%|███▌      | 7/20 [00:42<01:18,  6.04s/it]
 40%|████      | 8/20 [00:48<01:12,  6.02s/it]
 45%|████▌     | 9/20 [00:54<01:05,  5.98s/it]
 50%|█████     | 10/20 [01:00<01:00,  6.00s/it]
 55%|█████▌    | 11/20 [01:06<00:54,  6.04s/it]
 60%|██████    | 12/20 [01:12<00:48,  6.00s/it]
 65%|██████▌   | 13/20 [01:18<00:42,  6.08s/it]
 70%|███████   | 14/20 [01:24<00:36,  6.04s/it]
 75%|███████▌  | 15/20 [01:30<00:30,  6.10s/it]
 80%|████████  | 16/20 [01:36<00:24,  6.07s/it]
 85%|████████▌ | 17/20 [01:42<00:18,  6.03s/it]
 90%|█████████ | 18/20 [01:48<00:11,  5.99s/it]
 95%|█████████▌| 19/20 [01:54<00:05,  5.99s/it]
100%|██████████| 20/20 [02:00<00:00,  6.01s/it]
 50%|█████     | 10/20 [11:27<15:50, 95.02s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  50 % data.
```

```
  5%|█          | 1/20 [00:06<02:05,  6.60s/it]
 10%|█          | 2/20 [00:13<01:58,  6.56s/it]
 15%|█▌         | 3/20 [00:19<01:53,  6.66s/it]
 20%|██         | 4/20 [00:26<01:48,  6.77s/it]
 25%|██▌        | 5/20 [00:33<01:41,  6.76s/it]
 30%|███        | 6/20 [00:40<01:33,  6.69s/it]
 35%|███▌       | 7/20 [00:46<01:26,  6.64s/it]
 40%|████       | 8/20 [00:53<01:19,  6.61s/it]
 45%|████▌      | 9/20 [00:59<01:12,  6.58s/it]
 50%|█████      | 10/20 [01:06<01:06,  6.61s/it]
 55%|█████▌     | 11/20 [01:13<00:59,  6.60s/it]
 60%|██████     | 12/20 [01:19<00:53,  6.67s/it]
 65%|██████▌    | 13/20 [01:26<00:47,  6.73s/it]
 70%|███████    | 14/20 [01:33<00:39,  6.66s/it]
 75%|███████▌   | 15/20 [01:39<00:33,  6.67s/it]
 80%|████████   | 16/20 [01:46<00:26,  6.67s/it]
 85%|████████▌  | 17/20 [01:53<00:19,  6.67s/it]
 90%|█████████  | 18/20 [01:59<00:13,  6.64s/it]
 95%|█████████▌ | 19/20 [02:06<00:06,  6.65s/it]
100%|██████████| 20/20 [02:13<00:00,  6.60s/it]
 55%|█████▌     | 11/20 [13:40<15:57, 106.44s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  55 % data.

  5%|█          | 1/20 [00:07<02:20,  7.41s/it]
 10%|█          | 2/20 [00:15<02:15,  7.55s/it]
 15%|█▌         | 3/20 [00:22<02:07,  7.52s/it]
 20%|██         | 4/20 [00:29<01:58,  7.38s/it]
 25%|██▌        | 5/20 [00:36<01:49,  7.30s/it]
 30%|███        | 6/20 [00:44<01:41,  7.27s/it]
 35%|███▌       | 7/20 [00:51<01:34,  7.29s/it]
 40%|████       | 8/20 [00:58<01:26,  7.25s/it]
 45%|████▌      | 9/20 [01:06<01:20,  7.31s/it]
 50%|█████      | 10/20 [01:13<01:13,  7.36s/it]
 55%|█████▌     | 11/20 [01:20<01:05,  7.31s/it]
 60%|██████     | 12/20 [01:27<00:57,  7.24s/it]
 65%|██████▌    | 13/20 [01:34<00:50,  7.20s/it]
 70%|███████    | 14/20 [01:42<00:43,  7.23s/it]
 75%|███████▌   | 15/20 [01:49<00:36,  7.23s/it]
 80%|████████   | 16/20 [01:56<00:28,  7.23s/it]
 85%|████████▌  | 17/20 [02:04<00:21,  7.28s/it]
 90%|█████████  | 18/20 [02:11<00:14,  7.35s/it]
 95%|█████████▌ | 19/20 [02:18<00:07,  7.29s/it]
100%|██████████| 20/20 [02:25<00:00,  7.25s/it]
 60%|██████     | 12/20 [16:06<15:46, 118.29s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  60 % data.
```

```
  5%|▌         | 1/20 [00:07<02:25,  7.67s/it]
 10%|█         | 2/20 [00:15<02:18,  7.70s/it]
 15%|█▌        | 3/20 [00:23<02:11,  7.72s/it]
 20%|██        | 4/20 [00:30<02:03,  7.73s/it]
 25%|██▌       | 5/20 [00:38<01:57,  7.80s/it]
 30%|███       | 6/20 [00:46<01:50,  7.88s/it]
 35%|███▌      | 7/20 [00:54<01:42,  7.85s/it]
 40%|████      | 8/20 [01:02<01:34,  7.87s/it]
 45%|████▌     | 9/20 [01:10<01:26,  7.84s/it]
 50%|█████     | 10/20 [01:18<01:18,  7.81s/it]
 55%|█████▌    | 11/20 [01:25<01:10,  7.78s/it]
 60%|██████    | 12/20 [01:33<01:02,  7.76s/it]
 65%|██████▌   | 13/20 [01:41<00:54,  7.83s/it]
 70%|███████   | 14/20 [01:49<00:47,  7.88s/it]
 75%|███████▌  | 15/20 [01:57<00:39,  7.82s/it]
 80%|████████  | 16/20 [02:04<00:31,  7.76s/it]
 85%|████████▌ | 17/20 [02:12<00:23,  7.76s/it]
 90%|█████████ | 18/20 [02:20<00:15,  7.74s/it]
 95%|█████████▌| 19/20 [02:28<00:07,  7.78s/it]
100%|██████████| 20/20 [02:35<00:00,  7.72s/it]
 65%|██████▌   | 13/20 [18:42<15:06, 129.56s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  65 % data.

  5%|▌         | 1/20 [00:08<02:42,  8.55s/it]
 10%|█         | 2/20 [00:17<02:33,  8.55s/it]
 15%|█▌        | 3/20 [00:25<02:26,  8.59s/it]
 20%|██        | 4/20 [00:34<02:15,  8.50s/it]
 25%|██▌       | 5/20 [00:42<02:06,  8.43s/it]
 30%|███       | 6/20 [00:50<01:57,  8.37s/it]
 35%|███▌      | 7/20 [00:58<01:48,  8.34s/it]
 40%|████      | 8/20 [01:07<01:41,  8.42s/it]
 45%|████▌     | 9/20 [01:15<01:32,  8.45s/it]
 50%|█████     | 10/20 [01:24<01:23,  8.38s/it]
 55%|█████▌    | 11/20 [01:32<01:14,  8.31s/it]
 60%|██████    | 12/20 [01:40<01:06,  8.27s/it]
 65%|██████▌   | 13/20 [01:48<00:57,  8.27s/it]
 70%|███████   | 14/20 [01:56<00:49,  8.23s/it]
 75%|███████▌  | 15/20 [02:05<00:41,  8.32s/it]
 80%|████████  | 16/20 [02:13<00:33,  8.37s/it]
 85%|████████▌ | 17/20 [02:22<00:25,  8.35s/it]
 90%|█████████ | 18/20 [02:30<00:16,  8.30s/it]
 95%|█████████▌| 19/20 [02:38<00:08,  8.30s/it]
100%|██████████| 20/20 [02:46<00:00,  8.29s/it]
 70%|███████   | 14/20 [21:29<14:04, 140.80s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  70 % data.
```

```
  5%|▌         | 1/20 [00:08<02:47,  8.80s/it]
 10%|█         | 2/20 [00:17<02:39,  8.87s/it]
 15%|█▌        | 3/20 [00:26<02:31,  8.92s/it]
 20%|██        | 4/20 [00:35<02:22,  8.88s/it]
 25%|██▌       | 5/20 [00:44<02:13,  8.87s/it]
 30%|███       | 6/20 [00:53<02:03,  8.84s/it]
 35%|███▌      | 7/20 [01:02<01:55,  8.89s/it]
 40%|████      | 8/20 [01:11<01:46,  8.89s/it]
 45%|████▌     | 9/20 [01:20<01:38,  8.95s/it]
 50%|█████     | 10/20 [01:29<01:30,  9.00s/it]
 55%|█████▌    | 11/20 [01:38<01:20,  8.96s/it]
 60%|██████    | 12/20 [01:47<01:11,  8.91s/it]
 65%|██████▌   | 13/20 [01:56<01:02,  8.92s/it]
 70%|███████   | 14/20 [02:04<00:53,  8.90s/it]
 75%|███████▌  | 15/20 [02:13<00:44,  8.88s/it]
 80%|████████  | 16/20 [02:23<00:36,  9.04s/it]
 85%|████████▌ | 17/20 [02:32<00:27,  9.02s/it]
 90%|█████████ | 18/20 [02:40<00:17,  8.94s/it]
 95%|█████████▌| 19/20 [02:50<00:09,  9.07s/it]
100%|██████████| 20/20 [02:59<00:00,  9.02s/it]
 75%|███████▌  | 15/20 [24:28<12:41, 152.31s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  75 % data.

  5%|▌         | 1/20 [00:09<02:59,  9.42s/it]
 10%|█         | 2/20 [00:19<02:50,  9.50s/it]
 15%|█▌        | 3/20 [00:28<02:43,  9.59s/it]
 20%|██        | 4/20 [00:38<02:32,  9.54s/it]
 25%|██▌       | 5/20 [00:47<02:23,  9.56s/it]
 30%|███       | 6/20 [00:57<02:13,  9.54s/it]
 35%|███▌      | 7/20 [01:06<02:04,  9.55s/it]
 40%|████      | 8/20 [01:16<01:55,  9.58s/it]
 45%|████▌     | 9/20 [01:26<01:45,  9.63s/it]
 50%|█████     | 10/20 [01:35<01:35,  9.57s/it]
 55%|█████▌    | 11/20 [01:45<01:25,  9.48s/it]
 60%|██████    | 12/20 [01:54<01:15,  9.48s/it]
 65%|██████▌   | 13/20 [02:04<01:06,  9.51s/it]
 70%|███████   | 14/20 [02:13<00:57,  9.52s/it]
 75%|███████▌  | 15/20 [02:23<00:48,  9.64s/it]
 80%|████████  | 16/20 [02:33<00:38,  9.56s/it]
 85%|████████▌ | 17/20 [02:42<00:28,  9.52s/it]
 90%|█████████ | 18/20 [02:51<00:19,  9.52s/it]
 95%|█████████▌| 19/20 [03:01<00:09,  9.54s/it]
100%|██████████| 20/20 [03:10<00:00,  9.50s/it]
 80%|████████  | 16/20 [27:39<10:55, 163.91s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  80 % data.
```

```
  5%|▌         | 1/20 [00:10<03:15, 10.28s/it]
 10%|█         | 2/20 [00:20<03:05, 10.32s/it]
 15%|█▌        | 3/20 [00:30<02:53, 10.23s/it]
 20%|██        | 4/20 [00:40<02:42, 10.16s/it]
 25%|██▌       | 5/20 [00:50<02:31, 10.13s/it]
 30%|███       | 6/20 [01:00<02:21, 10.09s/it]
 35%|███▌      | 7/20 [01:11<02:11, 10.14s/it]
 40%|████      | 8/20 [01:21<02:02, 10.17s/it]
 45%|████▌     | 9/20 [01:31<01:50, 10.07s/it]
 50%|█████     | 10/20 [01:41<01:40, 10.06s/it]
 55%|█████▌    | 11/20 [01:51<01:31, 10.22s/it]
 60%|██████    | 12/20 [02:01<01:20, 10.12s/it]
 65%|██████▌   | 13/20 [02:11<01:11, 10.18s/it]
 70%|███████   | 14/20 [02:21<01:00, 10.15s/it]
 75%|███████▌  | 15/20 [02:31<00:50, 10.05s/it]
 80%|████████  | 16/20 [02:41<00:39, 10.00s/it]
 85%|████████▌ | 17/20 [02:51<00:30, 10.04s/it]
 90%|█████████ | 18/20 [03:01<00:20, 10.03s/it]
 95%|█████████▌| 19/20 [03:12<00:10, 10.12s/it]
100%|██████████| 20/20 [03:22<00:00, 10.10s/it]
 85%|████████▌ | 17/20 [31:02<08:46, 175.41s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  85 % data.

  5%|▌         | 1/20 [00:10<03:18, 10.42s/it]
 10%|█         | 2/20 [00:20<03:07, 10.43s/it]
 15%|█▌        | 3/20 [00:31<02:57, 10.46s/it]
 20%|██        | 4/20 [00:42<02:48, 10.52s/it]
 25%|██▌       | 5/20 [00:52<02:39, 10.62s/it]
 30%|███       | 6/20 [01:03<02:28, 10.59s/it]
 35%|███▌      | 7/20 [01:13<02:16, 10.54s/it]
 40%|████      | 8/20 [01:24<02:07, 10.61s/it]
 45%|████▌     | 9/20 [01:35<01:56, 10.62s/it]
 50%|█████     | 10/20 [01:46<01:46, 10.65s/it]
 55%|█████▌    | 11/20 [01:56<01:36, 10.71s/it]
 60%|██████    | 12/20 [02:07<01:25, 10.64s/it]
 65%|██████▌   | 13/20 [02:17<01:14, 10.64s/it]
 70%|███████   | 14/20 [02:28<01:03, 10.62s/it]
 75%|███████▌  | 15/20 [02:38<00:52, 10.57s/it]
 80%|████████  | 16/20 [02:49<00:42, 10.67s/it]
 85%|████████▌ | 17/20 [03:00<00:31, 10.65s/it]
 90%|█████████ | 18/20 [03:10<00:21, 10.60s/it]
 95%|█████████▌| 19/20 [03:21<00:10, 10.58s/it]
100%|██████████| 20/20 [03:32<00:00, 10.66s/it]
 90%|█████████ | 18/20 [34:34<06:13, 186.50s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  90 % data.
```

```
  5%|▌         | 1/20 [00:11<03:46, 11.94s/it]
 10%|█         | 2/20 [00:23<03:32, 11.78s/it]
 15%|█▌        | 3/20 [00:34<03:16, 11.56s/it]
 20%|██        | 4/20 [00:45<03:02, 11.41s/it]
 25%|██▌       | 5/20 [00:56<02:49, 11.32s/it]
 30%|███       | 6/20 [01:07<02:37, 11.22s/it]
 35%|███▌      | 7/20 [01:19<02:27, 11.36s/it]
 40%|████      | 8/20 [01:30<02:15, 11.32s/it]
 45%|████▌     | 9/20 [01:41<02:03, 11.20s/it]
 50%|█████     | 10/20 [01:52<01:51, 11.20s/it]
 55%|█████▌    | 11/20 [02:03<01:40, 11.19s/it]
 60%|██████    | 12/20 [02:15<01:29, 11.22s/it]
 65%|██████▌   | 13/20 [02:26<01:18, 11.27s/it]
 70%|███████   | 14/20 [02:37<01:07, 11.21s/it]
 75%|███████▌  | 15/20 [02:48<00:56, 11.23s/it]
 80%|████████  | 16/20 [03:00<00:44, 11.25s/it]
 85%|████████▌ | 17/20 [03:11<00:33, 11.31s/it]
 90%|█████████ | 18/20 [03:22<00:22, 11.33s/it]
 95%|█████████▌| 19/20 [03:33<00:11, 11.26s/it]
100%|██████████| 20/20 [03:45<00:00, 11.20s/it]
 95%|█████████▌| 19/20 [38:19<03:18, 198.07s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  95 % data.

  5%|▌         | 1/20 [00:11<03:43, 11.74s/it]
 10%|█         | 2/20 [00:23<03:31, 11.75s/it]
 15%|█▌        | 3/20 [00:35<03:22, 11.90s/it]
 20%|██        | 4/20 [00:47<03:09, 11.83s/it]
 25%|██▌       | 5/20 [00:58<02:55, 11.73s/it]
 30%|███       | 6/20 [01:10<02:44, 11.73s/it]
 35%|███▌      | 7/20 [01:22<02:32, 11.70s/it]
 40%|████      | 8/20 [01:35<02:25, 12.11s/it]
 45%|████▌     | 9/20 [01:47<02:12, 12.05s/it]
 50%|█████     | 10/20 [01:58<01:59, 11.90s/it]
 55%|█████▌    | 11/20 [02:10<01:46, 11.87s/it]
 60%|██████    | 12/20 [02:22<01:34, 11.80s/it]
 65%|██████▌   | 13/20 [02:34<01:23, 11.98s/it]
 70%|███████   | 14/20 [02:46<01:11, 11.90s/it]
 75%|███████▌  | 15/20 [02:58<00:59, 11.85s/it]
 80%|████████  | 16/20 [03:09<00:47, 11.85s/it]
 85%|████████▌ | 17/20 [03:21<00:35, 11.79s/it]
 90%|█████████ | 18/20 [03:33<00:23, 11.95s/it]
100%|██████████| 20/20 [03:57<00:00, 11.78s/it]
100%|██████████| 20/20 [42:16<00:00, 209.81s/it]

Finished Training for  100 % data.
```

Training using pre-trained weights.

In [0]:
```
from tqdm import tqdm
pretrained_res = {percentage: {'valid_loss': [], 'train_loss': [], 'valid_acc': [],
'train_acc': [],'test_acc': []} for percentage in percentages}
epochs = 20
live=False
# random initialisations
for percent in tqdm(pretrained_res.keys()):

  # build training data
  train_perc_loader = build_dataloader(trainxs_1, trainys_1, percentage=percent)
  trainxs_perc = train_perc_loader.dataset.X
  trainys_perc = train_perc_loader.dataset.Y
  validxs_tensor = validxs_tensor_1
  validys_tensor = validys_tensor_1
  testxs_tensor = testxs_tensor_1
  testys_tensor = testys_tensor_1

  #build model

  # define model, loss and optimiser
  cnn = CNN_class([28,28], [20,20,20], [[2,2],[2,2],[2,2]], [3,3,3], [128], 5)
  cnn.load_state_dict(torch.load('drive/My Drive/DL_Models/model_fin.pth'), strict=
False)

  cnn_weights = nn.Sequential(nn.Linear(180, 128), nn.ReLU(inplace=True), nn.Linear
(128, 5))
  cnn_weights.apply(init_weights)
  cnn.classif = cnn_weights


  if torch.cuda.is_available():
    cnn.cuda()

  loss = nn.CrossEntropyLoss()
  optimizer = optim.Adam(cnn.parameters(), lr=0.0001)

  train_loss = []
  valid_loss = []
  test_loss = []

  for epoch in tqdm(range(epochs)):

    running_loss = 0.0
    running_loss_without_reg = 0.0
    for i, data in enumerate(train_perc_loader, 0):
      inputs, labels = data

      if torch.cuda.is_available():
        inputs, labels = inputs.cuda(), labels.cuda()

      # zero the parameter gradients
      optimizer.zero_grad()

      # forward + backward + optimize
      outputs = cnn(inputs)
      loss_val_pre = loss(outputs, labels)
      regularisation = 0
      for param in cnn.parameters():
          regularisation += torch.norm(param, 2)
      loss_val = loss_val_pre + 0.01*regularisation
      loss_val.backward()
      optimizer.step()

    if torch.cuda.is_available():
```

```
  0%|          | 0/20 [00:00<?, ?it/s]
  0%|          | 0/20 [00:00<?, ?it/s]
  5%|▌         | 1/20 [00:00<00:15,  1.19it/s]
 10%|█         | 2/20 [00:01<00:14,  1.20it/s]
 15%|█▌        | 3/20 [00:02<00:14,  1.21it/s]
 20%|██        | 4/20 [00:03<00:13,  1.22it/s]
 25%|██▌       | 5/20 [00:04<00:12,  1.22it/s]
 30%|███       | 6/20 [00:04<00:11,  1.23it/s]
 35%|███▌      | 7/20 [00:05<00:10,  1.23it/s]
 40%|████      | 8/20 [00:06<00:09,  1.22it/s]
 45%|████▌     | 9/20 [00:07<00:08,  1.23it/s]
 50%|█████     | 10/20 [00:08<00:08,  1.23it/s]
 55%|█████▌    | 11/20 [00:08<00:07,  1.23it/s]
 60%|██████    | 12/20 [00:09<00:06,  1.23it/s]
 65%|██████▌   | 13/20 [00:10<00:05,  1.23it/s]
 70%|███████   | 14/20 [00:11<00:04,  1.24it/s]
 75%|███████▌  | 15/20 [00:12<00:04,  1.24it/s]
 80%|████████  | 16/20 [00:12<00:03,  1.24it/s]
 85%|████████▌ | 17/20 [00:13<00:02,  1.23it/s]
 90%|█████████ | 18/20 [00:14<00:01,  1.24it/s]
 95%|█████████▌| 19/20 [00:15<00:00,  1.24it/s]
100%|██████████| 20/20 [00:16<00:00,  1.24it/s]
  5%|▌         | 1/20 [00:16<05:08, 16.25s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  5 % data.

  5%|▌         | 1/20 [00:01<00:25,  1.35s/it]
 10%|█         | 2/20 [00:02<00:24,  1.36s/it]
 15%|█▌        | 3/20 [00:04<00:23,  1.36s/it]
 20%|██        | 4/20 [00:05<00:22,  1.39s/it]
 25%|██▌       | 5/20 [00:07<00:21,  1.45s/it]
 30%|███       | 6/20 [00:08<00:20,  1.44s/it]
 35%|███▌      | 7/20 [00:09<00:18,  1.41s/it]
 40%|████      | 8/20 [00:11<00:16,  1.41s/it]
 45%|████▌     | 9/20 [00:12<00:15,  1.40s/it]
 50%|█████     | 10/20 [00:14<00:14,  1.45s/it]
 55%|█████▌    | 11/20 [00:15<00:13,  1.46s/it]
 60%|██████    | 12/20 [00:17<00:11,  1.46s/it]
 65%|██████▌   | 13/20 [00:18<00:10,  1.49s/it]
 70%|███████   | 14/20 [00:20<00:08,  1.49s/it]
 75%|███████▌  | 15/20 [00:21<00:07,  1.47s/it]
 80%|████████  | 16/20 [00:23<00:05,  1.46s/it]
 85%|████████▌ | 17/20 [00:24<00:04,  1.46s/it]
 90%|█████████ | 18/20 [00:25<00:02,  1.45s/it]
 95%|█████████▌| 19/20 [00:27<00:01,  1.43s/it]
100%|██████████| 20/20 [00:28<00:00,  1.41s/it]
 10%|█         | 2/20 [00:45<06:00, 20.01s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  10 % data.
```

```
  5%|▌         | 1/20 [00:01<00:36,  1.95s/it]
 10%|█         | 2/20 [00:03<00:35,  1.95s/it]
 15%|█▌        | 3/20 [00:05<00:32,  1.94s/it]
 20%|██        | 4/20 [00:07<00:31,  1.94s/it]
 25%|██▌       | 5/20 [00:09<00:29,  1.95s/it]
 30%|███       | 6/20 [00:11<00:27,  1.95s/it]
 35%|███▌      | 7/20 [00:13<00:25,  1.96s/it]
 40%|████      | 8/20 [00:15<00:23,  1.96s/it]
 45%|████▌     | 9/20 [00:17<00:21,  1.97s/it]
 50%|█████     | 10/20 [00:19<00:19,  1.97s/it]
 55%|█████▌    | 11/20 [00:21<00:17,  1.97s/it]
 60%|██████    | 12/20 [00:23<00:15,  1.97s/it]
 65%|██████▌   | 13/20 [00:25<00:13,  1.97s/it]
 70%|███████   | 14/20 [00:27<00:11,  1.98s/it]
 75%|███████▌  | 15/20 [00:29<00:09,  1.97s/it]
 80%|████████  | 16/20 [00:31<00:07,  1.96s/it]
 85%|████████▌ | 17/20 [00:33<00:05,  1.95s/it]
 90%|█████████ | 18/20 [00:35<00:03,  1.95s/it]
 95%|█████████▌| 19/20 [00:37<00:02,  2.03s/it]
100%|██████████| 20/20 [00:39<00:00,  2.03s/it]
 15%|█▌        | 3/20 [01:24<07:19, 25.87s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  15 % data.

  5%|▌         | 1/20 [00:02<00:48,  2.53s/it]
 10%|█         | 2/20 [00:05<00:45,  2.54s/it]
 15%|█▌        | 3/20 [00:07<00:44,  2.63s/it]
 20%|██        | 4/20 [00:10<00:41,  2.60s/it]
 25%|██▌       | 5/20 [00:12<00:38,  2.58s/it]
 30%|███       | 6/20 [00:15<00:35,  2.56s/it]
 35%|███▌      | 7/20 [00:18<00:33,  2.56s/it]
 40%|████      | 8/20 [00:20<00:30,  2.54s/it]
 45%|████▌     | 9/20 [00:23<00:27,  2.53s/it]
 50%|█████     | 10/20 [00:25<00:25,  2.52s/it]
 55%|█████▌    | 11/20 [00:28<00:22,  2.52s/it]
 60%|██████    | 12/20 [00:30<00:20,  2.52s/it]
 65%|██████▌   | 13/20 [00:33<00:17,  2.54s/it]
 70%|███████   | 14/20 [00:35<00:15,  2.53s/it]
 75%|███████▌  | 15/20 [00:38<00:12,  2.54s/it]
 80%|████████  | 16/20 [00:40<00:10,  2.54s/it]
 85%|████████▌ | 17/20 [00:43<00:07,  2.55s/it]
 90%|█████████ | 18/20 [00:45<00:05,  2.56s/it]
 95%|█████████▌| 19/20 [00:48<00:02,  2.56s/it]
100%|██████████| 20/20 [00:51<00:00,  2.57s/it]
 20%|██        | 4/20 [02:15<08:55, 33.45s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  20 % data.
```

```
  5%|▌         | 1/20 [00:03<00:59,  3.11s/it]
 10%|█         | 2/20 [00:06<00:56,  3.16s/it]
 15%|█▌        | 3/20 [00:09<00:54,  3.19s/it]
 20%|██        | 4/20 [00:12<00:51,  3.19s/it]
 25%|██▌       | 5/20 [00:16<00:48,  3.25s/it]
 30%|███       | 6/20 [00:19<00:44,  3.20s/it]
 35%|███▌      | 7/20 [00:22<00:41,  3.17s/it]
 40%|████      | 8/20 [00:25<00:37,  3.16s/it]
 45%|████▌     | 9/20 [00:28<00:34,  3.15s/it]
 50%|█████     | 10/20 [00:31<00:31,  3.12s/it]
 55%|█████▌    | 11/20 [00:34<00:27,  3.10s/it]
 60%|██████    | 12/20 [00:37<00:24,  3.11s/it]
 65%|██████▌   | 13/20 [00:40<00:21,  3.10s/it]
 70%|███████   | 14/20 [00:44<00:18,  3.12s/it]
 75%|███████▌  | 15/20 [00:47<00:15,  3.13s/it]
 80%|████████  | 16/20 [00:50<00:12,  3.12s/it]
 85%|████████▌ | 17/20 [00:53<00:09,  3.11s/it]
 90%|█████████ | 18/20 [00:56<00:06,  3.13s/it]
 95%|█████████▌| 19/20 [00:59<00:03,  3.14s/it]
100%|██████████| 20/20 [01:02<00:00,  3.12s/it]
 25%|██▌       | 5/20 [03:18<10:34, 42.30s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  25 % data.

  5%|▌         | 1/20 [00:03<01:14,  3.93s/it]
 10%|█         | 2/20 [00:07<01:09,  3.86s/it]
 15%|█▌        | 3/20 [00:11<01:05,  3.83s/it]
 20%|██        | 4/20 [00:15<01:01,  3.85s/it]
 25%|██▌       | 5/20 [00:18<00:57,  3.81s/it]
 30%|███       | 6/20 [00:22<00:53,  3.81s/it]
 35%|███▌      | 7/20 [00:26<00:49,  3.80s/it]
 40%|████      | 8/20 [00:30<00:45,  3.76s/it]
 45%|████▌     | 9/20 [00:33<00:40,  3.73s/it]
 50%|█████     | 10/20 [00:37<00:37,  3.70s/it]
 55%|█████▌    | 11/20 [00:41<00:33,  3.71s/it]
 60%|██████    | 12/20 [00:44<00:29,  3.71s/it]
 65%|██████▌   | 13/20 [00:48<00:26,  3.71s/it]
 70%|███████   | 14/20 [00:52<00:22,  3.74s/it]
 75%|███████▌  | 15/20 [00:56<00:18,  3.73s/it]
 80%|████████  | 16/20 [00:59<00:14,  3.73s/it]
 85%|████████▌ | 17/20 [01:03<00:11,  3.78s/it]
 90%|█████████ | 18/20 [01:07<00:07,  3.76s/it]
 95%|█████████▌| 19/20 [01:11<00:03,  3.77s/it]
100%|██████████| 20/20 [01:15<00:00,  3.82s/it]
 30%|███       | 6/20 [04:33<12:10, 52.20s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  30 % data.
```

```
  5%|▌         | 1/20 [00:04<01:20,  4.22s/it]
 10%|█         | 2/20 [00:08<01:16,  4.24s/it]
 15%|█▌        | 3/20 [00:12<01:12,  4.25s/it]
 20%|██        | 4/20 [00:16<01:07,  4.24s/it]
 25%|██▌       | 5/20 [00:21<01:03,  4.24s/it]
 30%|███       | 6/20 [00:25<00:59,  4.27s/it]
 35%|███▌      | 7/20 [00:29<00:55,  4.30s/it]
 40%|████      | 8/20 [00:34<00:51,  4.29s/it]
 45%|████▌     | 9/20 [00:38<00:47,  4.31s/it]
 50%|█████     | 10/20 [00:42<00:43,  4.31s/it]
 55%|█████▌    | 11/20 [00:47<00:38,  4.29s/it]
 60%|██████    | 12/20 [00:51<00:35,  4.39s/it]
 65%|██████▌   | 13/20 [00:56<00:30,  4.36s/it]
 70%|███████   | 14/20 [01:00<00:26,  4.41s/it]
 75%|███████▌  | 15/20 [01:04<00:21,  4.37s/it]
 80%|████████  | 16/20 [01:09<00:17,  4.36s/it]
 85%|████████▌ | 17/20 [01:13<00:13,  4.39s/it]
 90%|█████████ | 18/20 [01:18<00:08,  4.39s/it]
 95%|█████████▌| 19/20 [01:22<00:04,  4.37s/it]
100%|██████████| 20/20 [01:26<00:00,  4.34s/it]
 35%|███▌      | 7/20 [06:00<13:32, 62.54s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  35 % data.

  5%|▌         | 1/20 [00:04<01:32,  4.86s/it]
 10%|█         | 2/20 [00:09<01:27,  4.87s/it]
 15%|█▌        | 3/20 [00:14<01:22,  4.88s/it]
 20%|██        | 4/20 [00:19<01:17,  4.87s/it]
 25%|██▌       | 5/20 [00:24<01:14,  4.99s/it]
 30%|███       | 6/20 [00:29<01:09,  4.98s/it]
 35%|███▌      | 7/20 [00:34<01:04,  4.98s/it]
 40%|████      | 8/20 [00:39<00:59,  4.96s/it]
 45%|████▌     | 9/20 [00:44<00:54,  4.91s/it]
 50%|█████     | 10/20 [00:49<00:48,  4.88s/it]
 55%|█████▌    | 11/20 [00:54<00:43,  4.85s/it]
 60%|██████    | 12/20 [00:58<00:38,  4.87s/it]
 65%|██████▌   | 13/20 [01:03<00:34,  4.87s/it]
 70%|███████   | 14/20 [01:08<00:29,  4.87s/it]
 75%|███████▌  | 15/20 [01:13<00:24,  4.87s/it]
 80%|████████  | 16/20 [01:18<00:19,  4.85s/it]
 85%|████████▌ | 17/20 [01:23<00:14,  4.94s/it]
 90%|█████████ | 18/20 [01:28<00:09,  4.92s/it]
 95%|█████████▌| 19/20 [01:33<00:04,  4.99s/it]
100%|██████████| 20/20 [01:38<00:00,  4.96s/it]
 40%|████      | 8/20 [07:39<14:39, 73.31s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  40 % data.
```

```
  5%|▌         | 1/20 [00:05<01:44,  5.53s/it]
 10%|█         | 2/20 [00:10<01:38,  5.48s/it]
 15%|█▌        | 3/20 [00:16<01:32,  5.45s/it]
 20%|██        | 4/20 [00:21<01:26,  5.43s/it]
 25%|██▌       | 5/20 [00:27<01:21,  5.42s/it]
 30%|███       | 6/20 [00:32<01:16,  5.43s/it]
 35%|███▌      | 7/20 [00:38<01:10,  5.46s/it]
 40%|████      | 8/20 [00:43<01:06,  5.57s/it]
 45%|████▌     | 9/20 [00:49<01:00,  5.53s/it]
 50%|█████     | 10/20 [00:55<00:55,  5.59s/it]
 55%|█████▌    | 11/20 [01:00<00:49,  5.54s/it]
 60%|██████    | 12/20 [01:05<00:44,  5.50s/it]
 65%|██████▌   | 13/20 [01:11<00:38,  5.46s/it]
 70%|███████   | 14/20 [01:16<00:32,  5.45s/it]
 75%|███████▌  | 15/20 [01:22<00:27,  5.47s/it]
 80%|████████  | 16/20 [01:27<00:22,  5.50s/it]
 85%|████████▌ | 17/20 [01:33<00:16,  5.49s/it]
 90%|█████████ | 18/20 [01:38<00:10,  5.47s/it]
 95%|█████████▌| 19/20 [01:44<00:05,  5.53s/it]
100%|██████████| 20/20 [01:49<00:00,  5.52s/it]
 45%|████▌     | 9/20 [09:28<15:27, 84.28s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  45 % data.

  5%|▌         | 1/20 [00:06<01:59,  6.28s/it]
 10%|█         | 2/20 [00:12<01:51,  6.21s/it]
 15%|█▌        | 3/20 [00:18<01:44,  6.14s/it]
 20%|██        | 4/20 [00:24<01:37,  6.10s/it]
 25%|██▌       | 5/20 [00:30<01:31,  6.07s/it]
 30%|███       | 6/20 [00:36<01:24,  6.07s/it]
 35%|███▌      | 7/20 [00:42<01:18,  6.04s/it]
 40%|████      | 8/20 [00:48<01:12,  6.04s/it]
 45%|████▌     | 9/20 [00:54<01:07,  6.13s/it]
 50%|█████     | 10/20 [01:00<01:00,  6.09s/it]
 55%|█████▌    | 11/20 [01:07<00:55,  6.18s/it]
 60%|██████    | 12/20 [01:13<00:48,  6.12s/it]
 65%|██████▌   | 13/20 [01:19<00:42,  6.06s/it]
 70%|███████   | 14/20 [01:25<00:36,  6.06s/it]
 75%|███████▌  | 15/20 [01:31<00:30,  6.12s/it]
 80%|████████  | 16/20 [01:37<00:24,  6.15s/it]
 85%|████████▌ | 17/20 [01:43<00:18,  6.12s/it]
 90%|█████████ | 18/20 [01:49<00:12,  6.08s/it]
 95%|█████████▌| 19/20 [01:55<00:06,  6.15s/it]
100%|██████████| 20/20 [02:02<00:00,  6.21s/it]
 50%|█████     | 10/20 [11:31<15:56, 95.69s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  50 % data.
```

```
  5%|▌         | 1/20 [00:06<02:04,  6.57s/it]
 10%|█         | 2/20 [00:13<01:58,  6.60s/it]
 15%|█▌        | 3/20 [00:19<01:51,  6.57s/it]
 20%|██        | 4/20 [00:26<01:45,  6.59s/it]
 25%|██▌       | 5/20 [00:32<01:38,  6.56s/it]
 30%|███       | 6/20 [00:39<01:31,  6.56s/it]
 35%|███▌      | 7/20 [00:45<01:25,  6.55s/it]
 40%|████      | 8/20 [00:52<01:19,  6.62s/it]
 45%|████▌     | 9/20 [00:59<01:13,  6.67s/it]
 50%|█████     | 10/20 [01:06<01:06,  6.66s/it]
 55%|█████▌    | 11/20 [01:12<00:59,  6.67s/it]
 60%|██████    | 12/20 [01:19<00:52,  6.60s/it]
 65%|██████▌   | 13/20 [01:25<00:45,  6.55s/it]
 70%|███████   | 14/20 [01:32<00:39,  6.59s/it]
 75%|███████▌  | 15/20 [01:39<00:33,  6.61s/it]
 80%|████████  | 16/20 [01:45<00:26,  6.58s/it]
 85%|████████▌ | 17/20 [01:52<00:20,  6.72s/it]
 90%|█████████ | 18/20 [01:59<00:13,  6.77s/it]
 95%|█████████▌| 19/20 [02:06<00:06,  6.72s/it]
100%|██████████| 20/20 [02:12<00:00,  6.69s/it]
 55%|█████▌    | 11/20 [13:44<16:01, 106.82s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  55 % data.

  5%|▌         | 1/20 [00:07<02:13,  7.03s/it]
 10%|█         | 2/20 [00:14<02:07,  7.08s/it]
 15%|█▌        | 3/20 [00:21<02:00,  7.08s/it]
 20%|██        | 4/20 [00:28<01:54,  7.14s/it]
 25%|██▌       | 5/20 [00:35<01:46,  7.13s/it]
 30%|███       | 6/20 [00:43<01:41,  7.28s/it]
 35%|███▌      | 7/20 [00:50<01:35,  7.34s/it]
 40%|████      | 8/20 [00:57<01:27,  7.26s/it]
 45%|████▌     | 9/20 [01:04<01:19,  7.19s/it]
 50%|█████     | 10/20 [01:11<01:11,  7.15s/it]
 55%|█████▌    | 11/20 [01:19<01:04,  7.16s/it]
 60%|██████    | 12/20 [01:26<00:57,  7.16s/it]
 65%|██████▌   | 13/20 [01:33<00:49,  7.13s/it]
 70%|███████   | 14/20 [01:40<00:43,  7.22s/it]
 75%|███████▌  | 15/20 [01:48<00:36,  7.31s/it]
 80%|████████  | 16/20 [01:55<00:29,  7.29s/it]
 85%|████████▌ | 17/20 [02:02<00:21,  7.26s/it]
 90%|█████████ | 18/20 [02:09<00:14,  7.21s/it]
 95%|█████████▌| 19/20 [02:16<00:07,  7.17s/it]
100%|██████████| 20/20 [02:24<00:00,  7.27s/it]
 60%|██████    | 12/20 [16:08<15:44, 118.11s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  60 % data.
```

```
  5%|▌         | 1/20 [00:07<02:31,  7.97s/it]
 10%|█         | 2/20 [00:15<02:23,  7.97s/it]
 15%|█▌        | 3/20 [00:24<02:16,  8.03s/it]
 20%|██        | 4/20 [00:31<02:07,  7.97s/it]
 25%|██▌       | 5/20 [00:39<01:58,  7.90s/it]
 30%|███       | 6/20 [00:47<01:50,  7.86s/it]
 35%|███▌      | 7/20 [00:55<01:42,  7.87s/it]
 40%|████      | 8/20 [01:03<01:34,  7.89s/it]
 45%|████▌     | 9/20 [01:11<01:26,  7.85s/it]
 50%|█████     | 10/20 [01:19<01:19,  7.95s/it]
 55%|█████▌    | 11/20 [01:27<01:12,  8.01s/it]
 60%|██████    | 12/20 [01:35<01:03,  7.93s/it]
 65%|██████▌   | 13/20 [01:42<00:55,  7.86s/it]
 70%|███████   | 14/20 [01:50<00:47,  7.86s/it]
 75%|███████▌  | 15/20 [01:58<00:39,  7.84s/it]
 80%|████████  | 16/20 [02:06<00:31,  7.84s/it]
 85%|████████▌ | 17/20 [02:14<00:23,  7.91s/it]
 90%|█████████ | 18/20 [02:22<00:16,  8.05s/it]
 95%|█████████▌| 19/20 [02:30<00:07,  7.98s/it]
100%|██████████| 20/20 [02:38<00:00,  7.91s/it]
 65%|██████▌   | 13/20 [18:46<15:11, 130.18s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  65 % data.

  5%|▌         | 1/20 [00:08<02:38,  8.34s/it]
 10%|█         | 2/20 [00:16<02:31,  8.42s/it]
 15%|█▌        | 3/20 [00:25<02:22,  8.40s/it]
 20%|██        | 4/20 [00:33<02:13,  8.36s/it]
 25%|██▌       | 5/20 [00:42<02:06,  8.44s/it]
 30%|███       | 6/20 [00:50<01:59,  8.54s/it]
 35%|███▌      | 7/20 [00:59<01:50,  8.51s/it]
 40%|████      | 8/20 [01:07<01:41,  8.43s/it]
 45%|████▌     | 9/20 [01:16<01:32,  8.41s/it]
 50%|█████     | 10/20 [01:24<01:24,  8.42s/it]
 55%|█████▌    | 11/20 [01:32<01:15,  8.40s/it]
 60%|██████    | 12/20 [01:41<01:08,  8.52s/it]
 65%|██████▌   | 13/20 [01:50<01:00,  8.58s/it]
 70%|███████   | 14/20 [01:58<00:51,  8.55s/it]
 75%|███████▌  | 15/20 [02:07<00:42,  8.47s/it]
 80%|████████  | 16/20 [02:15<00:33,  8.47s/it]
 85%|████████▌ | 17/20 [02:24<00:25,  8.50s/it]
 90%|█████████ | 18/20 [02:32<00:17,  8.54s/it]
 95%|█████████▌| 19/20 [02:41<00:08,  8.71s/it]
100%|██████████| 20/20 [02:50<00:00,  8.76s/it]
 70%|███████   | 14/20 [21:37<14:14, 142.37s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  70 % data.
```

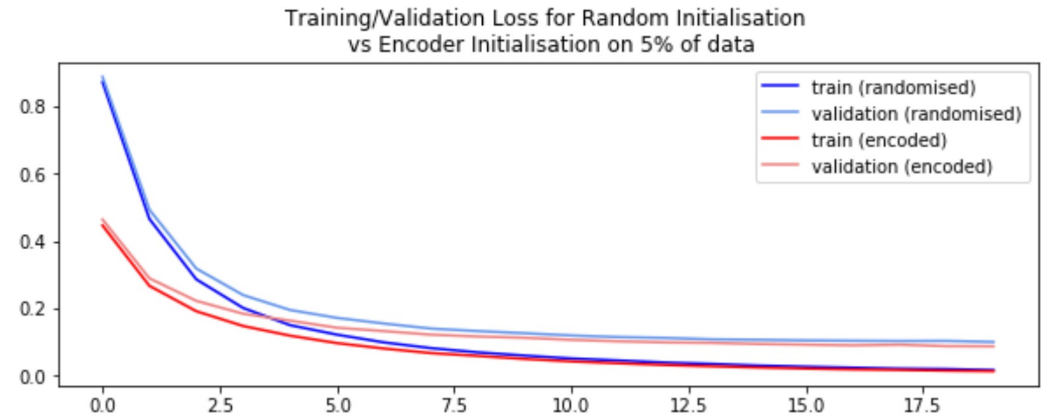
```
  5%|▎    | 1/20 [00:08<02:50,  8.98s/it]
 10%|▌    | 2/20 [00:17<02:41,  8.95s/it]
 15%|▊    | 3/20 [00:26<02:32,  8.96s/it]
 20%|█    | 4/20 [00:35<02:23,  8.96s/it]
 25%|█▎   | 5/20 [00:44<02:14,  8.99s/it]
 30%|█▌   | 6/20 [00:54<02:07,  9.12s/it]
 35%|█▊   | 7/20 [01:03<01:57,  9.05s/it]
 40%|██   | 8/20 [01:11<01:47,  8.96s/it]
 45%|██▎  | 9/20 [01:20<01:38,  8.96s/it]
 50%|██▌  | 10/20 [01:29<01:29,  8.92s/it]
 55%|██▊  | 11/20 [01:38<01:20,  8.91s/it]
 60%|███  | 12/20 [01:47<01:11,  8.99s/it]
 65%|███▎ | 13/20 [01:57<01:04,  9.15s/it]
 70%|███▌ | 14/20 [02:06<00:54,  9.10s/it]
 75%|███▊ | 15/20 [02:15<00:45,  9.03s/it]
 80%|████ | 16/20 [02:24<00:35,  8.98s/it]
 85%|████▎| 17/20 [02:32<00:26,  8.96s/it]
 90%|████▌| 18/20 [02:41<00:17,  8.90s/it]
 95%|████▊| 19/20 [02:50<00:09,  9.01s/it]
100%|█████| 20/20 [03:00<00:00,  9.09s/it]
 75%|███▊ | 15/20 [24:37<12:48, 153.74s/it]
  0%|     | 0/20 [00:00<?, ?it/s]

Finished Training for  75 % data.

  5%|▎    | 1/20 [00:09<02:59,  9.43s/it]
 10%|▌    | 2/20 [00:18<02:49,  9.41s/it]
 15%|▊    | 3/20 [00:28<02:41,  9.48s/it]
 20%|█    | 4/20 [00:37<02:32,  9.50s/it]
 25%|█▎   | 5/20 [00:47<02:23,  9.56s/it]
 30%|█▌   | 6/20 [00:57<02:15,  9.65s/it]
 35%|█▊   | 7/20 [01:07<02:04,  9.61s/it]
 40%|██   | 8/20 [01:16<01:54,  9.54s/it]
 45%|██▎  | 9/20 [01:26<01:45,  9.56s/it]
 50%|██▌  | 10/20 [01:35<01:35,  9.55s/it]
 55%|██▊  | 11/20 [01:45<01:26,  9.66s/it]
 60%|███  | 12/20 [01:56<01:19,  9.97s/it]
 65%|███▎ | 13/20 [02:05<01:08,  9.84s/it]
 70%|███▌ | 14/20 [02:15<00:58,  9.72s/it]
 75%|███▊ | 15/20 [02:24<00:48,  9.63s/it]
 80%|████ | 16/20 [02:34<00:38,  9.61s/it]
 85%|████▎| 17/20 [02:43<00:28,  9.55s/it]
 90%|████▌| 18/20 [02:53<00:19,  9.77s/it]
 95%|████▊| 19/20 [03:03<00:09,  9.70s/it]
100%|█████| 20/20 [03:13<00:00,  9.68s/it]
 80%|████ | 16/20 [27:51<11:02, 165.54s/it]
  0%|     | 0/20 [00:00<?, ?it/s]

Finished Training for  80 % data.
```

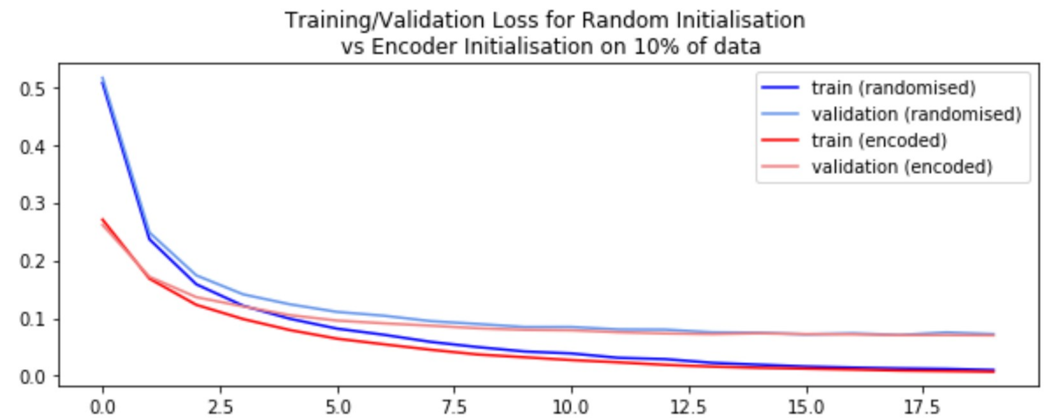
```
  5%|▌         | 1/20 [00:10<03:12, 10.15s/it]
 10%|█         | 2/20 [00:20<03:03, 10.17s/it]
 15%|█▌        | 3/20 [00:30<02:52, 10.13s/it]
 20%|██        | 4/20 [00:41<02:45, 10.35s/it]
 25%|██▌       | 5/20 [00:51<02:35, 10.35s/it]
 30%|███       | 6/20 [01:01<02:23, 10.27s/it]
 35%|███▌      | 7/20 [01:11<02:13, 10.25s/it]
 40%|████      | 8/20 [01:22<02:03, 10.27s/it]
 45%|████▌     | 9/20 [01:32<01:53, 10.33s/it]
 50%|█████     | 10/20 [01:43<01:43, 10.39s/it]
 55%|█████▌    | 11/20 [01:53<01:32, 10.32s/it]
 60%|██████    | 12/20 [02:03<01:21, 10.25s/it]
 65%|██████▌   | 13/20 [02:13<01:11, 10.25s/it]
 70%|███████   | 14/20 [02:23<01:01, 10.23s/it]
 75%|███████▌  | 15/20 [02:34<00:51, 10.37s/it]
 80%|████████  | 16/20 [02:45<00:41, 10.47s/it]
 85%|████████▌ | 17/20 [02:55<00:31, 10.38s/it]
 90%|█████████ | 18/20 [03:05<00:20, 10.29s/it]
 95%|█████████▌| 19/20 [03:15<00:10, 10.31s/it]
100%|██████████| 20/20 [03:26<00:00, 10.27s/it]
 85%|████████▌ | 17/20 [31:17<08:53, 177.71s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  85 % data.

  5%|▌         | 1/20 [00:11<03:31, 11.14s/it]
 10%|█         | 2/20 [00:22<03:22, 11.23s/it]
 15%|█▌        | 3/20 [00:33<03:10, 11.20s/it]
 20%|██        | 4/20 [00:44<02:57, 11.10s/it]
 25%|██▌       | 5/20 [00:55<02:45, 11.04s/it]
 30%|███       | 6/20 [01:06<02:34, 11.03s/it]
 35%|███▌      | 7/20 [01:17<02:24, 11.08s/it]
 40%|████      | 8/20 [01:28<02:11, 10.99s/it]
 45%|████▌     | 9/20 [01:39<01:59, 10.90s/it]
 50%|█████     | 10/20 [01:50<01:49, 10.93s/it]
 55%|█████▌    | 11/20 [02:00<01:38, 10.89s/it]
 60%|██████    | 12/20 [02:11<01:27, 10.93s/it]
 65%|██████▌   | 13/20 [02:23<01:16, 10.98s/it]
 70%|███████   | 14/20 [02:33<01:05, 10.96s/it]
 75%|███████▌  | 15/20 [02:44<00:54, 10.89s/it]
 80%|████████  | 16/20 [02:55<00:43, 10.87s/it]
 85%|████████▌ | 17/20 [03:06<00:32, 10.91s/it]
 90%|█████████ | 18/20 [03:17<00:22, 11.03s/it]
 95%|█████████▌| 19/20 [03:28<00:10, 10.94s/it]
100%|██████████| 20/20 [03:39<00:00, 10.85s/it]
 90%|█████████ | 18/20 [34:56<06:20, 190.18s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  90 % data.
```

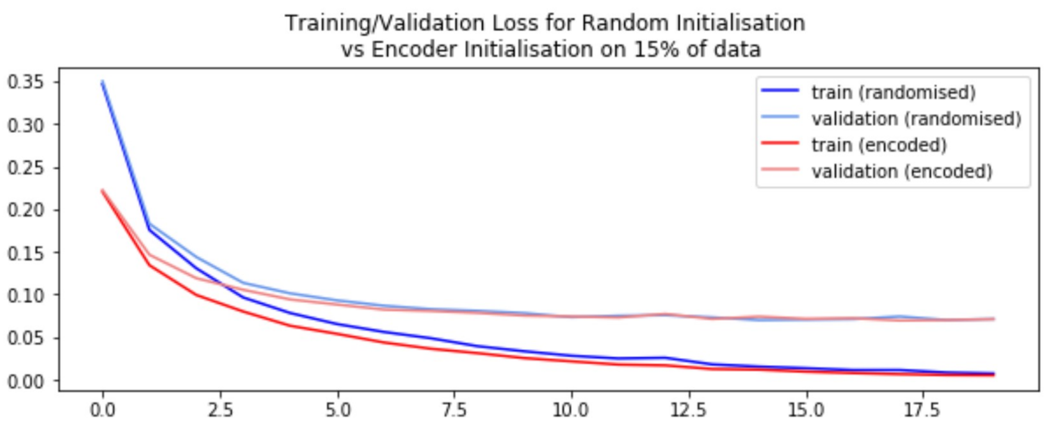
```
  5%|▌         | 1/20 [00:11<03:36, 11.41s/it]
 10%|█         | 2/20 [00:22<03:24, 11.37s/it]
 15%|█▌        | 3/20 [00:34<03:17, 11.62s/it]
 20%|██        | 4/20 [00:46<03:04, 11.56s/it]
 25%|██▌       | 5/20 [00:57<02:52, 11.50s/it]
 30%|███       | 6/20 [01:09<02:40, 11.46s/it]
 35%|███▌      | 7/20 [01:20<02:28, 11.43s/it]
 40%|████      | 8/20 [01:32<02:18, 11.51s/it]
 45%|████▌     | 9/20 [01:43<02:07, 11.55s/it]
 50%|█████     | 10/20 [01:55<01:55, 11.55s/it]
 55%|█████▌    | 11/20 [02:06<01:43, 11.54s/it]
 60%|██████    | 12/20 [02:18<01:32, 11.55s/it]
 65%|██████▌   | 13/20 [02:29<01:20, 11.55s/it]
 70%|███████   | 14/20 [02:41<01:09, 11.56s/it]
 75%|███████▌  | 15/20 [02:52<00:57, 11.46s/it]
 80%|████████  | 16/20 [03:03<00:45, 11.38s/it]
 85%|████████▌ | 17/20 [03:15<00:34, 11.36s/it]
 90%|█████████ | 18/20 [03:26<00:22, 11.40s/it]
 95%|█████████▌| 19/20 [03:38<00:11, 11.50s/it]
100%|██████████| 20/20 [03:49<00:00, 11.42s/it]
 95%|█████████▌| 19/20 [38:46<03:22, 202.05s/it]
  0%|          | 0/20 [00:00<?, ?it/s]

Finished Training for  95 % data.

  5%|▌         | 1/20 [00:11<03:46, 11.93s/it]
 10%|█         | 2/20 [00:23<03:33, 11.87s/it]
 15%|█▌        | 3/20 [00:35<03:20, 11.82s/it]
 20%|██        | 4/20 [00:47<03:11, 11.99s/it]
 25%|██▌       | 5/20 [00:59<02:58, 11.92s/it]
 30%|███       | 6/20 [01:11<02:46, 11.89s/it]
 35%|███▌      | 7/20 [01:23<02:34, 11.85s/it]
 40%|████      | 8/20 [01:34<02:21, 11.80s/it]
 45%|████▌     | 9/20 [01:47<02:11, 12.00s/it]
 50%|█████     | 10/20 [01:58<01:59, 11.90s/it]
 55%|█████▌    | 11/20 [02:10<01:46, 11.81s/it]
 60%|██████    | 12/20 [02:22<01:34, 11.85s/it]
 65%|██████▌   | 13/20 [02:34<01:22, 11.84s/it]
 70%|███████   | 14/20 [02:46<01:12, 12.03s/it]
 75%|███████▌  | 15/20 [02:58<00:59, 11.93s/it]
 80%|████████  | 16/20 [03:09<00:47, 11.81s/it]
 85%|████████▌ | 17/20 [03:22<00:35, 12.00s/it]
 90%|█████████ | 18/20 [03:34<00:23, 11.91s/it]
 95%|█████████▌| 19/20 [03:46<00:12, 12.12s/it]
100%|██████████| 20/20 [03:58<00:00, 12.03s/it]
100%|██████████| 20/20 [42:44<00:00, 213.00s/it]

Finished Training for  100 % data.
```

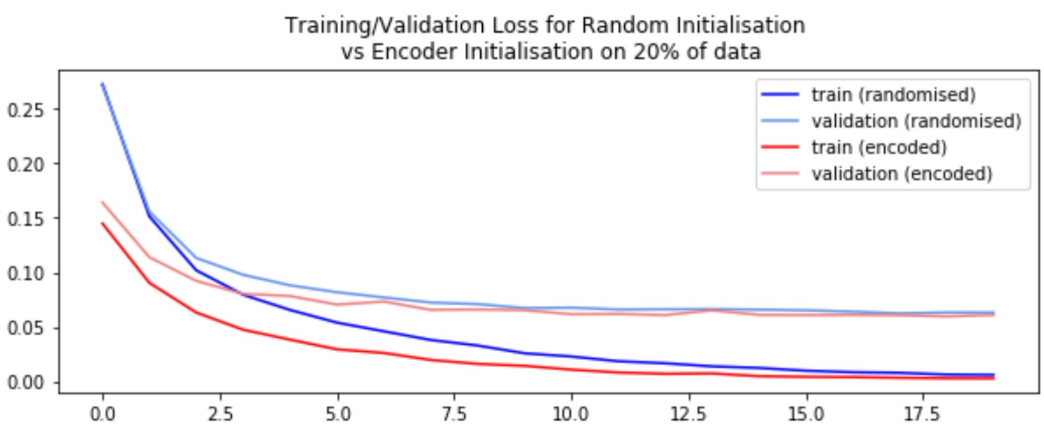
Plots of training and validation sets.

In [0]:
```
fig, axes = plt.subplots(20, 1, figsize=(10, 100))
fig.subplots_adjust(hspace=0.5)
for percent in random_res.keys():
  ax = axes[(percent//5 - 1)]
  train_loss_rand = random_res[percent]['train_loss']
  valid_loss_rand = random_res[percent]['valid_loss']
  train_loss_pretr = pretrained_res[percent]['train_loss']
  valid_loss_pretr = pretrained_res[percent]['valid_loss']
  ax.set_title('Training/Validation Loss for Random Initialisation \n vs Encoder In
itialisation on {}% of data'.format(percent))
  ax.plot(train_loss_rand, label='train (randomised)', c='b')
  ax.plot(valid_loss_rand, label='validation (randomised)', c='cornflowerblue')
  ax.plot(train_loss_pretr, label='train (encoded)', c='red')
  ax.plot(valid_loss_pretr, label='validation (encoded)', c='lightcoral')
  ax.legend()
```


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 5% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 10% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 15% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training/Validation Loss for Random Initialisation
vs Encoder Initialisation on 20% of data
train (randomised)
validation (randomised)
train (encoded)
validation (encoded)
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

The pre-trained initilisation could be said to be 'superior' when the size of the training data is smaller. This makes sense as the randomised weights do not have enough data to train on and therefore the pre training is an added bonus. This is due to the pre-training allowing the model to become 'familiarised' with the broad features of the distribution and takes away much of the overhead of the work. However, the only time that this benefit is really noticed is when training has not been running for a long time, and both types of initialisation converge to very similar accuracies. The pre-trained initialisation is always superior if you are unable to train for multiple epochs, but otherwise both initialisations share similar results.

5. Provide the final accuracy on the training, validation, and test set for the best model you obtained for each of the initialisation strategies.

In [0]:

```
print('Max random initialisation training accuracy {}'.format(max(random_res[10]['train_acc'])))
print('Max random initialisation validation accuracy {}'.format(max(random_res[100]['valid_acc'])))
print('Max random initialisation test accuracy {}'.format(max(random_res[100]['test_acc'])))
print('Max pre-trained initialisation train accuracy {}'.format(max(pretrained_res[10]['train_acc'])))
print('Max pre-trained initialisation validation accuracy {}'.format(max(pretrained_res[100]['valid_acc'])))
print('Max pre-trained initialisation test accuracy {}'.format(max(pretrained_res[95]['test_acc'])))
```

```
Max random initialisation training accuracy 100.0
Max random initialisation validation accuracy 98.94527363184079
Max random initialisation test accuracy 98.82
Max pre-trained initialisation train accuracy 100.0
Max pre-trained initialisation validation accuracy 98.98507462686568
Max pre-trained initialisation test accuracy 98.94
```