# Lab 4: Data Imputation using an Autoencoder

**Deadline**: Sunday, June 16, 9pm

**Late Penalty**: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted. Quercus submission time will be used, not your local computer time. You can submit your labs as many times as you want before the deadline, so please submit often and early.

**TA**: Huan Ling

In this lab, you will build and train an autoencoder to impute (or "fill in") missing data.

We will be using the Adult Data Set provided by the UCI Machine Learning Repository [1], available at https://archive.ics.uci.edu/ml/datasets/adult. The data set contains census record files of adults, including their age, martial status, the type of work they do, and other features.

Normally, people use this data set to build a supervised classification model to classify whether a person is a high income earner. We will not use the dataset for this original intended purpose.

Instead, we will perform the task of imputing (or "filling in") missing values in the dataset. For example, we may be missing one person's martial status, and another person's age, and a third person's level of education. Our model will predict the missing features based on the information that we do have about each person.

We will use a variation of a denoising autoencoder to solve this data imputation problem. Our autoencoder will be trained using inputs that have one categorical feature artificially removed, and the goal of the autoencoder is to correctly reconstruct all features, including the one removed from the input.

In the process, you are expected to learn to:

1. Clean and process continuous and categorical data for machine learning.
2. Implement an autoencoder that takes continuous and categorical (one-hot) inputs.
3. Tune the hyperparameters of an autoencoder.
4. Use baseline models to help interpret model performance.

[1] Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

### What to submit

Submit a PDF file containing all your code, outputs, and write-up. You can produce a PDF of your Google Colab file by going to File > Print and then save as PDF. The Colab instructions have more information.

Do not submit any other files produced by your code.

Include a link to your colab file in your submission.

# Colab Link

Include a link to your Colab file here. If you would like the TA to look at your Colab file in case your solutions are cut off, **please make sure that your Colab file is publicly accessible at the time of submission**.

Colab Link:https://drive.google.com/open?id=1kD7607FqmyEuzbiiwaDXd03YQEIFxzQ4

```
import csv
import numpy as np
import random
import torch
import torch.utils.data
import matplotlib.pyplot as plt
```

## Part 0

We will be using a package called `pandas` for this assignment.

If you are using Colab, `pandas` should already be available. If you are using your own computer, installation instructions for `pandas` are available here: https://pandas.pydata.org/pandas-docs/stable/install.html

```
import pandas as pd
```

# Part 1. Data Cleaning [15 pt]

The adult.data file is available at `https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data`

The function `pd.read_csv` loads the adult.data file into a pandas dataframe. You can read about the pandas documentation for `pd.read_csv` at https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

```
header = ['age', 'work', 'fnlwgt', 'edu', 'yredu', 'marriage', 'occupation',
 'relationship', 'race', 'sex', 'capgain', 'caploss', 'workhr', 'country']
df = pd.read_csv(
    "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data",
    names=header,
    index_col=False)
```

```
df.shape # there are 32561 rows (records) in the data frame, and 14 columns (features)
```

```
(32561, 14)
```

### Part (a) Continuous Features [3 pt]

For each of the columns `["age", "yredu", "capgain", "caploss", "workhr"]`, report the minimum, maximum, and average value across the dataset.

Then, normalize each of the features `["age", "yredu", "capgain", "caploss", "workhr"]` so that their values are always between 0 and 1. Make sure that you are actually modifying the dataframe

df.

Like numpy arrays and torch tensors, pandas data frames can be sliced. For example, we can display the first 3 rows of the data frame (3 records) below.

```
df[:3] # show the first 3 records
```

| | age | work | fnlwgt | edu | yredu | marriage | occupation | relationship | race |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 39 | State-gov | 77516 | Bachelors | 13 | Never-married | Adm-clerical | Not-in-family | White |
| **1** | 50 | Self-emp-not-inc | 83311 | Bachelors | 13 | Married-civ-spouse | Exec-managerial | Husband | White |

Alternatively, we can slice based on column names, for example df["race"], df["hr"], or even index multiple columns like below.

```
subdf = df[["age", "yredu", "capgain", "caploss", "workhr"]]
subdf[:3] # show the first 3 records
```

| | age | yredu | capgain | caploss | workhr |
|---|---|---|---|---|---|
| **0** | 39 | 13 | 2174 | 0 | 40 |
| **1** | 50 | 13 | 0 | 0 | 13 |
| **2** | 38 | 9 | 0 | 0 | 40 |

Numpy works nicely with pandas, like below:

```
np.sum(subdf["caploss"])
```

```
2842700
```

Just like numpy arrays, you can modify entire columns of data rather than one scalar element at a time. For example, the code

df["age"] = df["age"] + 1

would increment everyone's age by 1.

```
num_sample = df.shape[0]

for col in ["age", "yredu", "capgain", "caploss", "workhr"]:
  col_max = np.amax(df[col])
  col_min = np.amin(df[col])
  col_ave = np.sum(df[col])/num_sample
  print("The maximum "+col+ " is: " + str(col_max))
  print("The minimum "+col+ " is: " + str(col_min))
  print("The average "+col+ " is: " + str(col_ave))
  df[col] = (df[col] - col_min)/(col_max - col_min)
```

```
The maximum age is: 90
The minimum age is: 17
The average age is: 38.58164675532078
The maximum yredu is: 16
The minimum yredu is: 1
The average yredu is: 10.0806793403151
The maximum capgain is: 99999
The minimum capgain is: 0
The average capgain is: 1077.6488437087312
The maximum caploss is: 4356
The minimum caploss is: 0
The average caploss is: 87.303829734959
The maximum workhr is: 99
The minimum workhr is: 1
The average workhr is: 40.437455852092995
```

## Part (b) Categorical Features [1 pt]

What percentage of people in our data set are male? Note that the data labels all have an unfortunate space in the beginning, e.g. " Male" instead of "Male".

What percentage of people in our data set are female?

```
# hint: you can do something like this in pandas
sum(df["sex"] == " Male")
```

```
21790
```

```
num_sample = df.shape[0]
male_percentage = sum(df["sex"] == " Male") / num_sample
female_percentage = sum(df["sex"] == " Female") / num_sample

print(str(male_percentage*100)+ "% of the people are male.")
print(str(female_percentage*100)+ "% of the people are female.")
```

```
66.92054912318419% of the people are male.
33.07945087681583% of the people are female.
```

## Part (c) [2 pt]

Before proceeding, we will modify our data frame in a couple more ways:

1. We will restrict ourselves to using a subset of the features (to simplify our autoencoder)
2. We will remove any records (rows) already containing missing values, and store them in a second dataframe. We will only use records without missing values to train our autoencoder.

Both of these steps are done for you, below.

How many records contained missing features? What percentage of records were removed?

```
contcols = ["age", "yredu", "capgain", "caploss", "workhr"]
catcols = ["work", "marriage", "occupation", "edu", "relationship", "sex"]
features = contcols + catcols
df = df[features]
```

```
missing = pd.concat([df[c] == " ?" for c in catcols], axis=1).any(axis=1)
df_with_missing = df[missing]
df_not_missing = df[~missing]

num_with_missing = df_with_missing.shape[0]
removed_percentage = num_with_missing / df.shape[0]
print(str(num_with_missing) + " records contained missing features.")
print(str(removed_percentage*100)+ "% of records were removed")
```

```
1843 records contained missing features.
5.660145572924664% of records were removed
```

## Part (d) One-Hot Encoding [1 pt]

What are all the possible values of the feature "work" in `df_not_missing`? You may find the Python function `set` useful.

```
poss_val_work = set(df_not_missing["work"])
print(poss_val_work)
```

```
{' Without-pay', ' Self-emp-not-inc', ' Federal-gov', ' Local-gov', ' Self-emp-inc'
```

We will be using a one-hot encoding to represent each of the categorical variables. Our autoencoder will be trained using these one-hot encodings.

We will use the pandas function `get_dummies` to produce one-hot encodings for all of the categorical variables in `df_not_missing`.

```
data = pd.get_dummies(df_not_missing)
```

```
data[:3]
```

| | age | yredu | capgain | caploss | workhr | work_ Federal-gov | work_ Local-gov | work_ Private | work_ Self-emp-inc |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.301370 | 0.800000 | 0.02174 | 0.0 | 0.397959 | 0 | 0 | 0 | 0 |
| **1** | 0.452055 | 0.800000 | 0.00000 | 0.0 | 0.122449 | 0 | 0 | 0 | 0 |
| **2** | 0.287671 | 0.533333 | 0.00000 | 0.0 | 0.397959 | 0 | 0 | 1 | 0 |

## Part (e) One-Hot Encoding [2 pt]

The dataframe `data` contains the cleaned and normalized data that we will use to train our denoising autoencoder.

How many **columns** (features) are in the dataframe `data`?

Briefly explain where that number come from.

```
print("There are "+str(data.shape[1])+" columns in the dataframe data")
```

```
There are 57 columns in the dataframe data
```

The number is the second dimension of the data.

## Part (f) One-Hot Conversion [3 pt]

We will convert the pandas data frame `data` into numpy, so that it can be further converted into a PyTorch tensor. However, in doing so, we lose the column label information that a panda data frame automatically stores.

Complete the function `get_categorical_value` that will return the named value of a feature given a one-hot embedding. You may find the global variables `cat_index` and `cat_values` useful. (Display them and figure out what they are first.)

We will need this function in the next part of the lab to interpret our autoencoder outputs. So, the input to our function `get_categorical_values` might not actually be "one-hot" -- the input may instead contain real-valued predictions from our neural network.

```
datanp = data.values.astype(np.float32)
```

```
cat_index = {}  # Mapping of feature -> start index of feature in a record
cat_values = {} # Mapping of feature -> list of categorical values the feature can take

# build up the cat_index and cat_values dictionary
for i, header in enumerate(data.keys()):
    if "_" in header: # categorical header
        feature, value = header.split()
        feature = feature[:-1] # remove the last char; it is always an underscore
        if feature not in cat_index:
            cat_index[feature] = i
            cat_values[feature] = [value]
        else:
            cat_values[feature].append(value)


def get_onehot(record, feature):
    """
    Return the portion of `record` that is the one-hot encoding
    of `feature`. For example, since the feature "work" is stored
    in the indices [5:12] in each record, calling `get_range(record, "work")`
    is equivalent to accessing `record[5:12]`.

    Args:
        - record: a numpy array representing one record, formatted
                  the same way as a row in `data.np`
        - feature: a string, should be an element of `catcols`
    """
    start_index = cat_index[feature]
    stop_index = cat_index[feature] + len(cat_values[feature])
    return record[start_index:stop_index]

def get_categorical_value(onehot, feature):
    """
    Return the categorical value name of a feature given
```

```
    a one-hot vector representing the feature.

    Args:
        - onehot: a numpy array one-hot representation of the feature
        - feature: a string, should be an element of `catcols`

    Examples:

    >>> get_categorical_value(np.array([0., 0., 0., 0., 0., 1., 0.]), "work")
    'State-gov'
    >>> get_categorical_value(np.array([0.1, 0., 1.1, 0.2, 0., 1., 0.]), "work")
    'Private'
    """
    # <----- TODO: WRITE YOUR CODE HERE ----->
    # You may find the variables `cat_index` and `cat_values`
    # (created above) useful.
    return cat_values[feature][np.argmax(onehot)]


# more useful code, used during training, that depends on the function
# you write above

def get_feature(record, feature):
    """
    Return the categorical feature value of a record
    """
    onehot = get_onehot(record, feature)
    return get_categorical_value(onehot, feature)

def get_features(record):
    """
    Return a dictionary of all categorical feature values of a record
    """
    return { f: get_feature(record, f) for f in catcols }
```

### Part (g) Train/Test Split [3 pt]

Randomly split the data into approximately 70% training, 15% validation and 15% test.

Report the number of items in your training, validation, and test set.

```
# set the numpy seed for reproducibility
# https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.seed.html
np.random.seed(50)
np.random.shuffle(datanp)
train_split = int(len(datanp) * 0.7)
test_split = int(len(datanp) * 0.85)
train_data = datanp[:train_split]
val_data = datanp[train_split:test_split]
test_data = datanp[test_split:]
print("There are "+str(len(train_data))+" items in training set.")
print("There are "+str(len(val_data))+" items in validation set.")
print("There are "+str(len(test_data))+" items in test set.")
```

```
There are 21502 items in training set.
There are 4608 items in validation set.
There are 4608 items in test set.
```

## Part 2. Model Setup [5 pt]

### Part (a) [4 pt]

Design a fully-connected autoencoder by modifying the `encoder` and `decoder` below.

The input to this autoencoder will be the features of the `data`, with one categorical feature recorded as "missing". The output of the autoencoder should be the reconstruction of the same features, but with the missing value filled in.

**Note**: Do not reduce the dimensionality of the input too much! The output of your embedding is expected to contain information about ~11 features.

```
from torch import nn

class AutoEncoder(nn.Module):
    def __init__(self):
        self.name = "large"
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(57, 48),
            nn.ReLU(),
            nn.Linear(48, 24),
            nn.ReLU(),
            nn.Linear(24, 11)
        )
        self.decoder = nn.Sequential(
            nn.Linear(11, 24),
            nn.ReLU(),
            nn.Linear(24, 48),
            nn.ReLU(),
            nn.Linear(48, 57),
            nn.Sigmoid() # get to the range (0, 1)
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

### Part (b) [1 pt]

Explain why there is a sigmoid activation in the last step of the decoder.

(**Note**: the values inside the data frame `data` and the training code in Part 3 might be helpful.)

**The sigmoid activation makes it easier to compare the output with the acutal dataset. This is achieved through making the output have the same range as the actual dataset. All data in the dataset is either normalized to a value of 0 to 1 or has been encoded to a 0 or 1. A sigmoid gives us results in the same range (0 to 1).**

## Part 3. Training [18]

### Part (a) [6 pt]

We will train our autoencoder in the following way:

- In each iteration, we will hide one of the categorical features using the `zero_out_random_features` function

- We will pass the data with one missing feature through the autoencoder, and obtain a reconstruction
- We will check how close the reconstruction is compared to the original data -- including the value of the missing feature

Complete the code to train the autoencoder, and plot the training and validation loss every few iterations. You may also want to plot training and validation "accuracy" every few iterations, as we will define in part (b). You may also want to checkpoint your model every few iterations or epochs.

Use `nn.MSELoss()` as your loss function. (Side note: you might recognize that this loss function is not ideal for this problem, but we will use it anyway.)

```
def zero_out_feature(records, feature):
    """ Set the feature missing in records, by setting the appropriate
    columns of records to 0
    """
    start_index = cat_index[feature]
    stop_index = cat_index[feature] + len(cat_values[feature])
    records[:, start_index:stop_index] = 0
    return records

def zero_out_random_feature(records):
    """ Set one random feature missing in records, by setting the
    appropriate columns of records to 0
    """
    return zero_out_feature(records, random.choice(catcols))

def train(model, train_data, val_data, batch_size = 64, num_epochs=5, learning_rate=1e-4):
    """ Training loop. You should update this."""
    ######################
    train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, num_work
    val_loader = torch.utils.data.DataLoader(val_data, batch_size=batch_size, num_workers=
    ######################
    torch.manual_seed(42)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    ######################
    train_acc = np.zeros(num_epochs)
    train_loss = np.zeros(num_epochs)
    val_acc = np.zeros(num_epochs)
    val_loss = np.zeros(num_epochs)
    iters = []
    ######################
    for epoch in range(num_epochs):
      total_loss = 0.0
      i = 0

      for data in train_loader:
        datam = zero_out_random_feature(data.clone()) # zero out one categorical feature
        recon = model(datam)
        loss = criterion(recon, data)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        total_loss += loss.item()
        i += 1
      iters.append(epoch + 1)
      train_loss[epoch] = float(total_loss) / i
      val_loss[epoch] = get_loss(model, val_loader, criterion)
      train_acc[epoch] = get_accuracy(model, train_loader)
      val_acc[epoch] = get_accuracy(model, val_loader)
      print(("Epoch {}: Train acc: {}, Train loss: {} |" + "Validation acc: {}, Validation
                  epoch + 1, train_acc[epoch], train_loss[epoch], val_acc[epoch], val_los
      model_path = get_model_name(model.name, batch_size, learning_rate, epoch)
```

```
      torch.save(model.state_dict(), model_path)

    ######################
    #plotting
    plt.title("Train vs. Validation Loss")
    plt.plot(iters, train_loss, label = "Train")
    plt.plot(iters, val_loss, label = "Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend(loc='best')
    plt.show()

    plt.title("Train vs. Validation Accuracy")
    plt.plot(iters, train_acc, label = "Train")
    plt.plot(iters, val_acc, label = "Validation")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend(loc='best')
    plt.show()

    val_max = np.amax(val_acc)
    max_idx = np.argmax(val_acc)
    print("Final Training Accuracy: {}" .format(train_acc[-1]))
    print("Final Validation Accuracy: {}".format(val_acc[-1]))
    print("Highest Validation Accuracy: {} at epoch {}".format(val_max, max_idx+1))

def get_loss(model, data_loader, criterion):
    i = 0
    total_loss = 0.0
    for data in data_loader:
      datam = zero_out_random_feature(data.clone()) # zero out one categorical feature
      recon = model(datam)
      loss = criterion(recon, data)
      total_loss += loss.item()
      i += 1
    return float(total_loss) / i

def get_model_name(name, batch_size, learning_rate, epoch):

    path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(name, batch_size,
                                                   learning_rate, epoch)
    return path
```

## Part (b) [3 pt]

While plotting training and validation loss is valuable, loss values are harder to compare than accuracy percentages. It would be nice to have a measure of "accuracy" in this problem.

Since we will only be imputing missing categorical values, we will define an accuracy measure. For each record and for each categorical feature, we determine whether the model can predict the categorical feature given all the other features of the record.

A function `get_accuracy` is written for you. It is up to you to figure out how to use the function. **You don't need to submit anything in this part.** To earn the marks, correctly plot the training and validation accuracy every few iterations as part of your training curve.

```
def get_accuracy(model, data_loader):
    """Return the "accuracy" of the autoencoder model across a data set.
    That is, for each record and for each categorical feature,
    we determine whether the model can successfully predict the value
    of the categorical feature given all the other features of the
    record. The returned "accuracy" measure is the percentage of times
    that our model is successful.
```

```
    Args:
       - model: the autoencoder model, an instance of nn.Module
       - data_loader: an instance of torch.utils.data.DataLoader

    Example (to illustrate how get_accuracy is intended to be called.
             Depending on your variable naming this code might require
             modification.)

        >>> model = AutoEncoder()
        >>> vdl = torch.utils.data.DataLoader(data_valid, batch_size=256, shuffle=True)
        >>> get_accuracy(model, vdl)
    """
    total = 0
    acc = 0
    for col in catcols:
        for item in data_loader: # minibatches
            inp = item.detach().numpy()
            out = model(zero_out_feature(item.clone(), col)).detach().numpy()
            for i in range(out.shape[0]): # record in minibatch
                acc += int(get_feature(out[i], col) == get_feature(inp[i], col))
                total += 1
    return acc / total
```

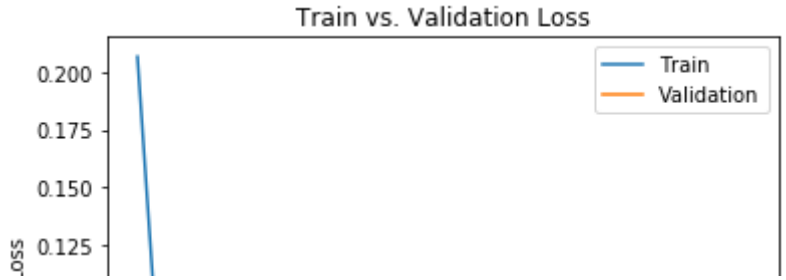
## Part (c) [4 pt]

Run your updated training code, using reasonable initial hyperparameters.

Include your training curve in your submission.

```
auto_encoder = AutoEncoder()
train(auto_encoder, train_data, val_data, batch_size = 64, num_epochs=30, learning_rate=0.
```

```
Epoch 1: Train acc: 0.40593898242023996, Train loss: 0.20652450685433688 |Validatio
Epoch 2: Train acc: 0.45905032090038134, Train loss: 0.07408521304439221 |Validatio
Epoch 3: Train acc: 0.46171673952810593, Train loss: 0.07068785220118505 |Validatio
Epoch 4: Train acc: 0.462375592968096, Train loss: 0.06984966186185677 |Validation
Epoch 5: Train acc: 0.48562924379127526, Train loss: 0.06727465109101363 |Validatio
Epoch 6: Train acc: 0.5413992496822001, Train loss: 0.058571472974671496 |Validatio
Epoch 7: Train acc: 0.5519098998542771, Train loss: 0.055401449147168366 |Validatio
Epoch 8: Train acc: 0.5610718382786097, Train loss: 0.054449239246813316 |Validatio
Epoch 9: Train acc: 0.5697842061203608, Train loss: 0.053764281184634284 |Validatio
Epoch 10: Train acc: 0.5697764549034199, Train loss: 0.053215638180041595 |Validati
Epoch 11: Train acc: 0.5704043034756456, Train loss: 0.052596658401723416 |Validati
Epoch 12: Train acc: 0.5722180882398537, Train loss: 0.05212981066489149 |Validatio
Epoch 13: Train acc: 0.5756518773447431, Train loss: 0.0514106470204535 |Validation
Epoch 14: Train acc: 0.5784500666604657, Train loss: 0.050760179480892564 |Validati
Epoch 15: Train acc: 0.5809072024307816, Train loss: 0.05000390590257233 |Validatio
Epoch 16: Train acc: 0.5812715096270115, Train loss: 0.04957844897927273 |Validatio
Epoch 17: Train acc: 0.5806669147056088, Train loss: 0.04865708593500867 |Validatio
Epoch 18: Train acc: 0.5806049049700803, Train loss: 0.04787040698075933 |Validatio
Epoch 19: Train acc: 0.5839146746039128, Train loss: 0.04697686835147795 |Validatio
Epoch 20: Train acc: 0.5906117260409884, Train loss: 0.04598424665718561 |Validatio
Epoch 21: Train acc: 0.5942160419185812, Train loss: 0.044686497909770834 |Validati
Epoch 22: Train acc: 0.595998821815025, Train loss: 0.043233683554544336 |Validatio
Epoch 23: Train acc: 0.6009828543081264, Train loss: 0.04179002553047169 |Validatio
Epoch 24: Train acc: 0.6049359749480668, Train loss: 0.040498130389356186 |Validati
Epoch 25: Train acc: 0.6076566520943788, Train loss: 0.03943393643324574 |Validatio
Epoch 26: Train acc: 0.609501441726351, Train loss: 0.03815626292045982 |Validation
Epoch 27: Train acc: 0.6081449787616656, Train loss: 0.03735228265923936 |Validatio
Epoch 28: Train acc: 0.6055328186525284, Train loss: 0.036501261788154285 |Validati
Epoch 29: Train acc: 0.6018199857377609, Train loss: 0.035733632904122625 |Validati
Epoch 30: Train acc: 0.6044011409791338, Train loss: 0.03523693888426004 |Validatio
```




### Part (d) [5 pt]

Tune your hyperparameters, training at least 4 different models (4 sets of hyperparameters).

Do not include all your training curves. Instead, explain what hyperparameters you tried, what their effect was, and what your thought process was as you chose the next set of hyperparameters to try.

**My tunning strategy is to tune one hyperparameter at a time and decide the optimal value for that hyperparameter by observing its effects on the training curve.**

**I will focus on tuning the learning rate first because it has significant effects on validation loss compared to other hyperparameters such as batch size. With a large learning rate, the model is able to learn really fast but it also produces high validation loss. While a small learning rate allows the model to learn more accurately with a**

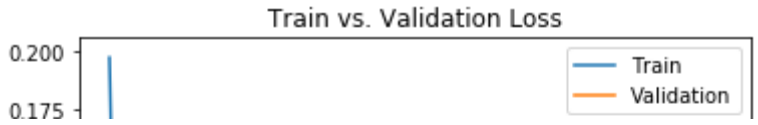
**longer training time. However, it does not mean that the smaller the learning rate, the higher the validation accuracy. The optimal learning rate is the one that produces the highest validation accuracy while taking moderate time to train. My first goal is to find the optimal learning rate. My next step is to find the best batch_size based on the learning rate that I have chosen. Batch_size usually has a smaller offect on the validation accuracy. It is used to adjust the noise in the training curve and speed up the training process. The batch_size will be chosen based on the same criteria -- validation accuracy.**

**In the process of hyperparameter search, I will keep my num_eopch high all the time to make sure the model always overfit. Adjustments about num_epoch will be made at the end to avoid overfitting.**

```
#set1
#batch_size = 64, num_epochs=100, learning_rate=0.0001
#For the first set, I will first use a high num_epochs to make it overfit and
#see how the model performs
net1 = AutoEncoder()
train(net1, train_data, val_data, batch_size = 64, num_epochs=100, learning_rate=0.0001)
```

```
Epoch 1: Train acc: 0.4028850029454624, Train loss: 0.19725065323568525 |Validation
Epoch 2: Train acc: 0.45928285740861313, Train loss: 0.07387299449848277 |Validatio
Epoch 3: Train acc: 0.45758534089852104, Train loss: 0.07093040762646567 |Validatio
Epoch 4: Train acc: 0.45917434037143834, Train loss: 0.07038541375437662 |Validatio
Epoch 5: Train acc: 0.46303444640808605, Train loss: 0.06973548947523038 |Validatio
Epoch 6: Train acc: 0.4690183858865842, Train loss: 0.0677887378371365 |Validation
Epoch 7: Train acc: 0.5441974389979227, Train loss: 0.05930189309375627 |Validation
Epoch 8: Train acc: 0.5625058134127058, Train loss: 0.05321010104602292 |Validation
Epoch 9: Train acc: 0.5578085759464236, Train loss: 0.050497336368564336 |Validatio
Epoch 10: Train acc: 0.5593588193346355, Train loss: 0.04925370255174736 |Validatio
Epoch 11: Train acc: 0.5601881995473289, Train loss: 0.048439653956198266 |Validati
Epoch 12: Train acc: 0.5649862028338449, Train loss: 0.04783289359572033 |Validatio
Epoch 13: Train acc: 0.5721173224196199, Train loss: 0.04704417062125036 |Validatio
Epoch 14: Train acc: 0.5720863175518557, Train loss: 0.04631447369631912 |Validatio
Epoch 15: Train acc: 0.573163736706663, Train loss: 0.045532495116016695 |Validatio
Epoch 16: Train acc: 0.5775276718444796, Train loss: 0.04472537312124457 |Validatio
Epoch 17: Train acc: 0.5817443338604161, Train loss: 0.044355808646373805 |Validati
Epoch 18: Train acc: 0.583550367407683, Train loss: 0.043705305561334604 |Validatio
Epoch 19: Train acc: 0.5857749666697671, Train loss: 0.043169914678271325 |Validati
Epoch 20: Train acc: 0.5875577465662108, Train loss: 0.04255816916979495 |Validatio
Epoch 21: Train acc: 0.5881313366198493, Train loss: 0.04192257263431592 |Validatio
Epoch 22: Train acc: 0.5906969894273401, Train loss: 0.04113563451738585 |Validatio
Epoch 23: Train acc: 0.5930998666790687, Train loss: 0.04022932922955425 |Validatio
Epoch 24: Train acc: 0.6001767277462562, Train loss: 0.03909434911440171 |Validatio
Epoch 25: Train acc: 0.6050367407683006, Train loss: 0.03827597099977235 |Validatio
Epoch 26: Train acc: 0.606261433044988, Train loss: 0.03748398226508427 |Validation
Epoch 27: Train acc: 0.6088193346355378, Train loss: 0.03723628590038667 |Validatio
Epoch 28: Train acc: 0.6070753108237993, Train loss: 0.03663114142892439 |Validatio
Epoch 29: Train acc: 0.6056180820388801, Train loss: 0.036361064174256864 |Validati
Epoch 30: Train acc: 0.6067962670139212, Train loss: 0.03602950584830805 |Validatio
Epoch 31: Train acc: 0.6078581837348463, Train loss: 0.035824664984829724 |Validati
Epoch 32: Train acc: 0.6097882367531703, Train loss: 0.035435014470879524 |Validati
Epoch 33: Train acc: 0.610013022044461, Train loss: 0.03530139951146252 |Validation
Epoch 34: Train acc: 0.6081217251108424, Train loss: 0.035095138098334984 |Validati
Epoch 35: Train acc: 0.6125476699841875, Train loss: 0.03501149342351016 |Validatio
Epoch 36: Train acc: 0.606245930611106, Train loss: 0.03475257427808607 |Validation
Epoch 37: Train acc: 0.6167333271323597, Train loss: 0.034498202215347974 |Validati
Epoch 38: Train acc: 0.6135243233187611, Train loss: 0.03420892467034892 |Validatio
Epoch 39: Train acc: 0.6086178029950702, Train loss: 0.03421691529053662 |Validatio
Epoch 40: Train acc: 0.6154078690354385, Train loss: 0.03399004509472953 |Validatio
Epoch 41: Train acc: 0.6141831767587511, Train loss: 0.03358905244663003 |Validatio
Epoch 42: Train acc: 0.6180975413139863, Train loss: 0.03355380843415679 |Validatio
Epoch 43: Train acc: 0.6177254829008154, Train loss: 0.0333735116790714 |Validation
Epoch 44: Train acc: 0.6161287322109571, Train loss: 0.03315326466690749 |Validatio
Epoch 45: Train acc: 0.6194617554956128, Train loss: 0.03299587568090785 |Validatio
Epoch 46: Train acc: 0.6226397544414473, Train loss: 0.03277370971738405 |Validatio
Epoch 47: Train acc: 0.6224847301026261, Train loss: 0.03266397065904346 |Validatio
Epoch 48: Train acc: 0.6186401264998604, Train loss: 0.032530051844549324 |Validati
Epoch 49: Train acc: 0.6237481784640189, Train loss: 0.032222165210571675 |Validati
Epoch 50: Train acc: 0.6214693206833473, Train loss: 0.03213439144504567 |Validatio
Epoch 51: Train acc: 0.623515641955787, Train loss: 0.031918498881471656 |Validatio
Epoch 52: Train acc: 0.6247790903171798, Train loss: 0.03165831961225541 |Validatio
Epoch 53: Train acc: 0.624019471056956, Train loss: 0.031701049756347424 |Validatio
Epoch 54: Train acc: 0.625321675503054, Train loss: 0.03128627435459445 |Validation
Epoch 55: Train acc: 0.6257014851331659, Train loss: 0.031059886184742765 |Validati
Epoch 56: Train acc: 0.6238101881995474, Train loss: 0.030865522278916268 |Validati
Epoch 57: Train acc: 0.6253681828047003, Train loss: 0.03062587688189177 |Validatio
```

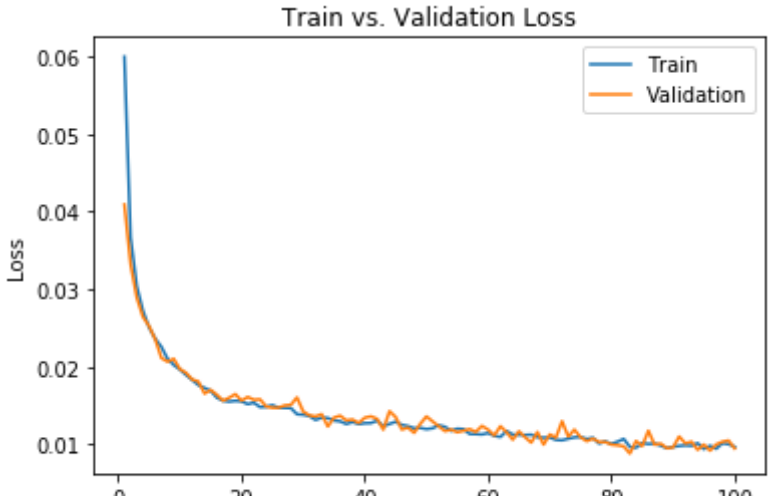
```
Epoch 58: Train acc: 0.6239807149722506, Train loss: 0.030498281963879152 |Validati
Epoch 59: Train acc: 0.6246628220630639, Train loss: 0.030239686159239637 |Validati
Epoch 60: Train acc: 0.6238644467181348, Train loss: 0.03009609752778141 |Validatio
Epoch 61: Train acc: 0.6262440703190401, Train loss: 0.029775699517423555 |Validati
Epoch 62: Train acc: 0.6236629150776672, Train loss: 0.02962662695374872 |Validatio
Epoch 63: Train acc: 0.6229265494682665, Train loss: 0.02949872239315439 |Validatio
Epoch 64: Train acc: 0.6221049204725142, Train loss: 0.029257539826046144 |Validati
Epoch 65: Train acc: 0.624810095184944, Train loss: 0.029149004247128255 |Validatio
Epoch 66: Train acc: 0.6243295197345984, Train loss: 0.02884360559151641 |Validatio
Epoch 67: Train acc: 0.6249806219576474, Train loss: 0.028656913327895814 |Validati
Epoch 68: Train acc: 0.6261355532818652, Train loss: 0.028573040972419438 |Validati
Epoch 69: Train acc: 0.6234768858710817, Train loss: 0.028373980800443815 |Validati
Epoch 70: Train acc: 0.6257479924348123, Train loss: 0.028104949171566182 |Validati
Epoch 71: Train acc: 0.6238566955011937, Train loss: 0.02803829356673218 |Validatio
Epoch 72: Train acc: 0.6242675099990699, Train loss: 0.0277343324296886 |Validation
Epoch 73: Train acc: 0.6243605246023626, Train loss: 0.02772120759999823 |Validatio
Epoch 74: Train acc: 0.6230273152885003, Train loss: 0.027560230710410645 |Validati
Epoch 75: Train acc: 0.6222366911605122, Train loss: 0.027287029328622987 |Validati
Epoch 76: Train acc: 0.62843766471336, Train loss: 0.027494213144694055 |Validation
Epoch 77: Train acc: 0.618833906923387, Train loss: 0.027384203903022267 |Validatio
Epoch 78: Train acc: 0.625445694974111, Train loss: 0.02725095851790337 |Validation
Epoch 79: Train acc: 0.6238644467181348, Train loss: 0.026884842037578068 |Validati
Epoch 80: Train acc: 0.6266858896846805, Train loss: 0.027121308996963005 |Validati
Epoch 81: Train acc: 0.6230970762409699, Train loss: 0.026956398283974045 |Validati
Epoch 82: Train acc: 0.622725017827799, Train loss: 0.0267346861184619 |Validation
Epoch 83: Train acc: 0.6213297987784082, Train loss: 0.02667720593689453 |Validatio
Epoch 84: Train acc: 0.6230350665054414, Train loss: 0.026540443134893264 |Validati
Epoch 85: Train acc: 0.6244767928564785, Train loss: 0.02625651378184557 |Validatio
Epoch 86: Train acc: 0.6214848231172294, Train loss: 0.02664497178713126 |Validatio
Epoch 87: Train acc: 0.6241899978296592, Train loss: 0.026272090102013732 |Validati
Epoch 88: Train acc: 0.6257014851331659, Train loss: 0.02631971505027087 |Validatio
Epoch 89: Train acc: 0.625709236350107, Train loss: 0.02614393874093713 |Validation
Epoch 90: Train acc: 0.6231048274579108, Train loss: 0.026083523163660652 |Validati
Epoch 91: Train acc: 0.6269339286267944, Train loss: 0.026203014161659495 |Validati
Epoch 92: Train acc: 0.6261355532818652, Train loss: 0.025924876603918772 |Validati
Epoch 93: Train acc: 0.62441478312095, Train loss: 0.026046713747616326 |Validation
Epoch 94: Train acc: 0.6237171735962546, Train loss: 0.025708404003775547 |Validati
Epoch 95: Train acc: 0.6252829194183487, Train loss: 0.025917681870937702 |Validati
Epoch 96: Train acc: 0.6275850308498434, Train loss: 0.02550518236100851 |Validatio
Epoch 97: Train acc: 0.6247480854494155, Train loss: 0.025617229918550168 |Validati
Epoch 98: Train acc: 0.624678324496946, Train loss: 0.0256463687789316 |Validation
Epoch 99: Train acc: 0.6261898118004526, Train loss: 0.02540163963013107 |Validatio
Epoch 100: Train acc: 0.6222754472452176, Train loss: 0.025222440477504972 |Validat
```

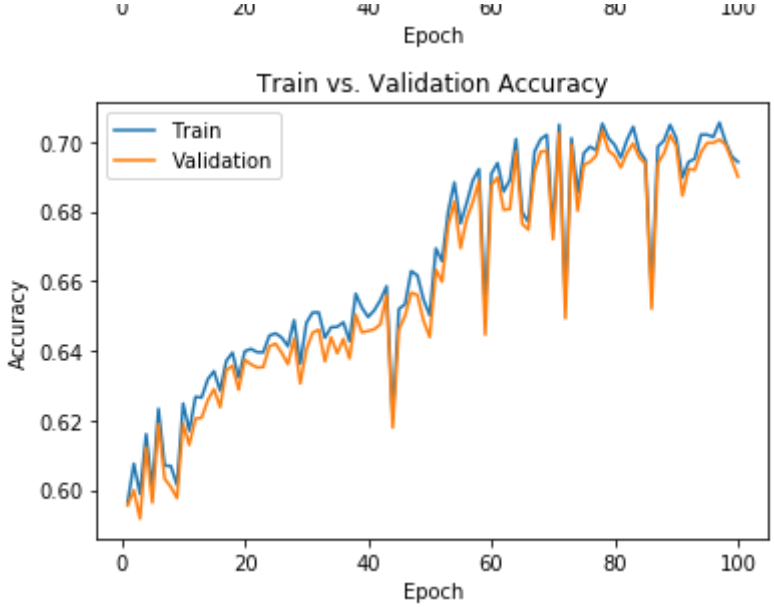



```
#set2
#batch_size = 64, num_epochs=100, learning_rate=0.005
#It seems like the model learns really slow, and the validation accuracy does
#not increase much after 25 epochs. So I decided to use a large learning rate
#to speed up the training process. My other intention is to see whether a
#smaller learning rate will produce a smaller validation loss. If that is true,
#I will keep decreasing the learning rate until the model does not perform any
#better or worse.
net2 = AutoEncoder()
train(net2, train_data, val_data, batch_size = 64, num_epochs=100, learning_rate=0.005)
```

```
Epoch 1: Train acc: 0.596386382662078, Train loss: 0.05993588466108555 |Validation
Epoch 2: Train acc: 0.6075016277555576, Train loss: 0.03667531706326242 |Validation
Epoch 3: Train acc: 0.5988280159985118, Train loss: 0.030406505290773652 |Validatio
Epoch 4: Train acc: 0.615989210306018, Train loss: 0.027242500285085822 |Validation
Epoch 5: Train acc: 0.6004247666883701, Train loss: 0.02508612493762658 |Validation
Epoch 6: Train acc: 0.6233063590983784, Train loss: 0.023641388372717693 |Validatio
Epoch 7: Train acc: 0.6069900474374477, Train loss: 0.022573025029573944 |Validatio
Epoch 8: Train acc: 0.6068427743155675, Train loss: 0.021042164955620786 |Validatio
Epoch 9: Train acc: 0.6014169224568258, Train loss: 0.0202497256686911 |Validation
Epoch 10: Train acc: 0.624810095184944, Train loss: 0.019661477899977138 |Validatio
Epoch 11: Train acc: 0.6169658636405916, Train loss: 0.018964844093369765 |Validati
Epoch 12: Train acc: 0.6266936409016216, Train loss: 0.018317930242379328 |Validati
Epoch 13: Train acc: 0.6265308653458593, Train loss: 0.0176566241037411 |Validation
Epoch 14: Train acc: 0.6317784392149568, Train loss: 0.017285015719521436 |Validati
Epoch 15: Train acc: 0.6340960530803336, Train loss: 0.0169274982486275 |Validation
Epoch 16: Train acc: 0.6285074256658295, Train loss: 0.016030480586258427 |Validati
Epoch 17: Train acc: 0.6370492667348774, Train loss: 0.01564743624268366 |Validatio
Epoch 18: Train acc: 0.6394288903357828, Train loss: 0.0155331847823358 |Validation
Epoch 19: Train acc: 0.6322280097975382, Train loss: 0.015626857603257077 |Validati
Epoch 20: Train acc: 0.6398242023997768, Train loss: 0.015564939812369024 |Validati
Epoch 21: Train acc: 0.640498558273649, Train loss: 0.015210061074910863 |Validatio
Epoch 22: Train acc: 0.6395606610237807, Train loss: 0.015394659384453138 |Validati
Epoch 23: Train acc: 0.6395219049390755, Train loss: 0.014821731260911162 |Validati
Epoch 24: Train acc: 0.6443431618764146, Train loss: 0.014851366201626314 |Validati
Epoch 25: Train acc: 0.6449555080147583, Train loss: 0.015069662917604936 |Validati
Epoch 26: Train acc: 0.643754069388894, Train loss: 0.01474343775494379 |Validation
Epoch 27: Train acc: 0.6412659287508139, Train loss: 0.014725554196740546 |Validati
Epoch 28: Train acc: 0.6487536043158776, Train loss: 0.014697184628208302 |Validati
Epoch 29: Train acc: 0.6362353889560661, Train loss: 0.013901719359204262 |Validati
Epoch 30: Train acc: 0.6480017362725948, Train loss: 0.013841329440419074 |Validati
Epoch 31: Train acc: 0.6509471987101975, Train loss: 0.013675896468893847 |Validati
Epoch 32: Train acc: 0.6509859547949028, Train loss: 0.013170117612129875 |Validati
Epoch 33: Train acc: 0.6436843084364245, Train loss: 0.013454441381673817 |Validati
Epoch 34: Train acc: 0.646738287911202, Train loss: 0.013416032524435736 |Validatio
Epoch 35: Train acc: 0.6468235512975538, Train loss: 0.01308968517280716 |Validatio
Epoch 36: Train acc: 0.648149009394475, Train loss: 0.012993574623362206 |Validatio
Epoch 37: Train acc: 0.6427386599696152, Train loss: 0.01264693292268064 |Validatio
Epoch 38: Train acc: 0.656342045701175, Train loss: 0.012898116903324123 |Validatio
Epoch 39: Train acc: 0.6523501689765293, Train loss: 0.012640693823673896 |Validati
Epoch 40: Train acc: 0.6496759991318637, Train loss: 0.012746006959960574 |Validati
Epoch 41: Train acc: 0.6515595448485412, Train loss: 0.012767172982594707 |Validati
Epoch 42: Train acc: 0.6545050072861439, Train loss: 0.013080161319868196 |Validati
Epoch 43: Train acc: 0.6584968840107897, Train loss: 0.012351300735082034 |Validati
Epoch 44: Train acc: 0.6207872135925341, Train loss: 0.012649357018138593 |Validati
Epoch 45: Train acc: 0.6519781105633584, Train loss: 0.012938668609075691 |Validati
Epoch 46: Train acc: 0.6533500759619261, Train loss: 0.012528540131392046 |Validati
Epoch 47: Train acc: 0.6628763215824884, Train loss: 0.012371324530769405 |Validati
Epoch 48: Train acc: 0.6616128732210957, Train loss: 0.012095136775834752 |Validati
Epoch 49: Train acc: 0.6546600316249651, Train loss: 0.012147130983759694 |Validati
Epoch 50: Train acc: 0.6501953306669147, Train loss: 0.011985213724463912 |Validati
Epoch 51: Train acc: 0.6693873438129786, Train loss: 0.012097066309901752 |Validati
Epoch 52: Train acc: 0.6657287694167985, Train loss: 0.012470324207762522 |Validati
Epoch 53: Train acc: 0.6799755061544662, Train loss: 0.012280404955769578 |Validati
Epoch 54: Train acc: 0.6883313180169287, Train loss: 0.011762556576979392 |Validati
Epoch 55: Train acc: 0.676580473134282, Train loss: 0.012034937491296745 |Validatio
Epoch 56: Train acc: 0.6824636467925465, Train loss: 0.011983064606153806 |Validati
Epoch 57: Train acc: 0.6889204105044492, Train loss: 0.011366046826532554 |Validati
```

```
Epoch 58: Train acc: 0.6921604191858122, Train loss: 0.011350396855656678 |Validati
Epoch 59: Train acc: 0.6518618423092426, Train loss: 0.011302119014241422 |Validati
Epoch 60: Train acc: 0.6908194586550088, Train loss: 0.011477153440604784 |Validati
Epoch 61: Train acc: 0.6939044429975506, Train loss: 0.01112141611104432 |Validatio
Epoch 62: Train acc: 0.6856959042569684, Train loss: 0.010975600078901542 |Validati
Epoch 63: Train acc: 0.6890444299755062, Train loss: 0.011740280033942932 |Validati
Epoch 64: Train acc: 0.7008030260750938, Train loss: 0.01127877907406184 |Validatio
Epoch 65: Train acc: 0.6797972281648219, Train loss: 0.01124949789872127 |Validatio
Epoch 66: Train acc: 0.6769990388490993, Train loss: 0.011229190789717472 |Validati
Epoch 67: Train acc: 0.6971909589805599, Train loss: 0.011269068667648494 |Validati
Epoch 68: Train acc: 0.7007875236412117, Train loss: 0.010948900019727825 |Validati
Epoch 69: Train acc: 0.7020199671348402, Train loss: 0.01088165576636259 |Validatio
Epoch 70: Train acc: 0.6746349176820761, Train loss: 0.010910752187815629 |Validati
Epoch 71: Train acc: 0.7049266734877375, Train loss: 0.010604137764035147 |Validati
Epoch 72: Train acc: 0.6539391684494466, Train loss: 0.010554241429504362 |Validati
Epoch 73: Train acc: 0.7010588162341488, Train loss: 0.010749685374321416 |Validati
Epoch 74: Train acc: 0.6854401140979134, Train loss: 0.010922317303040819 |Validati
Epoch 75: Train acc: 0.6967336371810374, Train loss: 0.010968754244025885 |Validati
Epoch 76: Train acc: 0.6986094316807738, Train loss: 0.010715288250052947 |Validati
Epoch 77: Train acc: 0.697594022261495, Train loss: 0.010863586513247961 |Validatio
Epoch 78: Train acc: 0.705228970948439, Train loss: 0.010133337613383663 |Validatio
Epoch 79: Train acc: 0.7010820698849719, Train loss: 0.010403057497959318 |Validati
Epoch 80: Train acc: 0.6991287632158248, Train loss: 0.010086536472607847 |Validati
Epoch 81: Train acc: 0.6954469351688215, Train loss: 0.010358532365422607 |Validati
Epoch 82: Train acc: 0.7005394846990978, Train loss: 0.010724657016994786 |Validati
Epoch 83: Train acc: 0.7043220785663349, Train loss: 0.00969280462664929 |Validatio
Epoch 84: Train acc: 0.6974002418379686, Train loss: 0.00959095387259454 |Validatio
Epoch 85: Train acc: 0.6947493256441262, Train loss: 0.010190286456019663 |Validati
Epoch 86: Train acc: 0.6567528601990512, Train loss: 0.010077661826341813 |Validati
Epoch 87: Train acc: 0.6986404365485381, Train loss: 0.01012863050259295 |Validatio
Epoch 88: Train acc: 0.7003844603602766, Train loss: 0.009857143046766785 |Validati
Epoch 89: Train acc: 0.7048879174030322, Train loss: 0.009530241461193543 |Validati
Epoch 90: Train acc: 0.7010045577155614, Train loss: 0.009703411322940761 |Validati
Epoch 91: Train acc: 0.6897575419340837, Train loss: 0.009822692509346996 |Validati
Epoch 92: Train acc: 0.6942532477598983, Train loss: 0.009891668381030849 |Validati
Epoch 93: Train acc: 0.6951368864911791, Train loss: 0.009842711577047816 |Validati
Epoch 94: Train acc: 0.7020587232195454, Train loss: 0.010170993407622777 |Validati
Epoch 95: Train acc: 0.7020664744364865, Train loss: 0.009414464348256366 |Validati
Epoch 96: Train acc: 0.7013921185626143, Train loss: 0.009847410429280163 |Validati
Epoch 97: Train acc: 0.7055312684091403, Train loss: 0.009461738769979482 |Validati
Epoch 98: Train acc: 0.6999813970793415, Train loss: 0.010162824885420767 |Validati
Epoch 99: Train acc: 0.6957879887142281, Train loss: 0.010053530882855523 |Validati
Epoch 100: Train acc: 0.6942377453260162, Train loss: 0.009698466638274979 |Validat
```



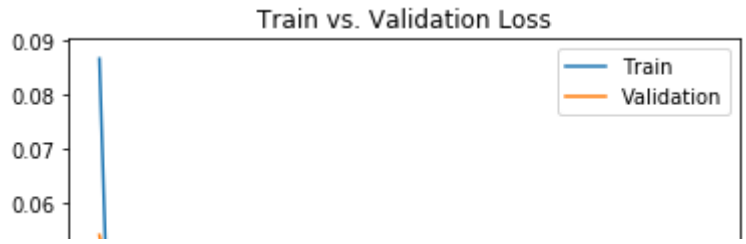



```
Final Training Accuracy: 0.6942377453260162
Final Validation Accuracy: 0.6900318287037037
Highest Validation Accuracy: 0.7032335069444444 at epoch 78
```

```
#set3
#batch_size = 64, num_epochs=100, learning_rate=0.001
#The second outperforms the first model by having a high validtion accuracy.
#I will decrease the learning rate further more to see if it produces better
#results.
net3 = AutoEncoder()
train(net3, train_data, val_data, batch_size = 64, num_epochs=100, learning_rate=0.001)
```

```
Epoch 1: Train acc: 0.551375065885344, Train loss: 0.08664687695203438 |Validation
Epoch 2: Train acc: 0.5823411775648777, Train loss: 0.05111284673746143 |Validation
Epoch 3: Train acc: 0.5936889591665891, Train loss: 0.045585290961233635 |Validatio
Epoch 4: Train acc: 0.5892785167271262, Train loss: 0.04141842667013407 |Validation
Epoch 5: Train acc: 0.6008588348370694, Train loss: 0.03950798068018187 |Validation
Epoch 6: Train acc: 0.6048042042600689, Train loss: 0.03792476784327023 |Validation
Epoch 7: Train acc: 0.6087573249000093, Train loss: 0.03563107986424473 |Validation
Epoch 8: Train acc: 0.6032074535702105, Train loss: 0.03400971056405632 |Validation
Epoch 9: Train acc: 0.5966654264719561, Train loss: 0.03248777879690308 |Validation
Epoch 10: Train acc: 0.6053545406628841, Train loss: 0.030551388067015933 |Validati
Epoch 11: Train acc: 0.6033392242582085, Train loss: 0.028623758876208393 |Validati
Epoch 12: Train acc: 0.6112842216227948, Train loss: 0.027577693385648586 |Validati
Epoch 13: Train acc: 0.6094006759061172, Train loss: 0.026710350321428405 |Validati
Epoch 14: Train acc: 0.612477909031718, Train loss: 0.02563245738634751 |Validation
Epoch 15: Train acc: 0.6167798344340062, Train loss: 0.024811003523479615 |Validati
Epoch 16: Train acc: 0.6127569528415961, Train loss: 0.02342340731549831 |Validatio
Epoch 17: Train acc: 0.6103075682882213, Train loss: 0.02287416795401701 |Validatio
Epoch 18: Train acc: 0.6092999100858835, Train loss: 0.022310182635140206 |Validati
Epoch 19: Train acc: 0.6148420301987412, Train loss: 0.02157715434441343 |Validatio
Epoch 20: Train acc: 0.6069977986543887, Train loss: 0.021016078934605633 |Validati
Epoch 21: Train acc: 0.6048429603447741, Train loss: 0.02061916301525863 |Validatio
Epoch 22: Train acc: 0.609485939292469, Train loss: 0.020277713087298685 |Validatio
Epoch 23: Train acc: 0.6074473692369702, Train loss: 0.019690496555995196 |Validati
Epoch 24: Train acc: 0.6076489008774377, Train loss: 0.01922161972504996 |Validatio
Epoch 25: Train acc: 0.6080364617244908, Train loss: 0.01894064540032386 |Validatio
Epoch 26: Train acc: 0.6107028803522153, Train loss: 0.018759371172304133 |Validati
Epoch 27: Train acc: 0.6060521501875794, Train loss: 0.018641257763928956 |Validati
Epoch 28: Train acc: 0.6057498527268781, Train loss: 0.018460584444082564 |Validati
Epoch 29: Train acc: 0.6042461166403126, Train loss: 0.01819432660981658 |Validatio
Epoch 30: Train acc: 0.6151830837441479, Train loss: 0.0176139781972216 |Validation
Epoch 31: Train acc: 0.6088658419371842, Train loss: 0.017376595975032876 |Validati
Epoch 32: Train acc: 0.6120283384491365, Train loss: 0.017411822791300005 |Validati
Epoch 33: Train acc: 0.6159349517874306, Train loss: 0.01695908943656832 |Validatio
Epoch 34: Train acc: 0.6138343719964034, Train loss: 0.017092036632155732 |Validati
Epoch 35: Train acc: 0.610540104796453, Train loss: 0.016636223138664804 |Validatio
Epoch 36: Train acc: 0.6080597153753139, Train loss: 0.016699235675679075 |Validati
Epoch 37: Train acc: 0.6100440269122253, Train loss: 0.0163528637023687 |Validation
Epoch 38: Train acc: 0.616105478560134, Train loss: 0.016258428776858465 |Validatio
Epoch 39: Train acc: 0.615206337394971, Train loss: 0.016057458447037442 |Validatio
Epoch 40: Train acc: 0.6141909279756922, Train loss: 0.015504887513261997 |Validati
Epoch 41: Train acc: 0.6116640312529067, Train loss: 0.015598268360398444 |Validati
Epoch 42: Train acc: 0.6135553281865253, Train loss: 0.015950615551056608 |Validati
Epoch 43: Train acc: 0.616756580783183, Train loss: 0.015538097697398848 |Validatio
Epoch 44: Train acc: 0.6195857749666698, Train loss: 0.015180262211721302 |Validati
Epoch 45: Train acc: 0.6172139025827055, Train loss: 0.015376306621224753 |Validati
Epoch 46: Train acc: 0.6195625213158465, Train loss: 0.015413421874178485 |Validati
Epoch 47: Train acc: 0.6165783027935385, Train loss: 0.0155447562130922 |Validation
Epoch 48: Train acc: 0.6155008836387312, Train loss: 0.015228387736161017 |Validati
Epoch 49: Train acc: 0.6111679533686789, Train loss: 0.01535702851279417 |Validatio
Epoch 50: Train acc: 0.6155783958081419, Train loss: 0.015208037309570327 |Validati
Epoch 51: Train acc: 0.6163302638514246, Train loss: 0.015165669183867673 |Validati
Epoch 52: Train acc: 0.6181828047003379, Train loss: 0.014983371123283481 |Validati
Epoch 53: Train acc: 0.6110206802467988, Train loss: 0.015043923351359331 |Validati
Epoch 54: Train acc: 0.6180200291445757, Train loss: 0.015222644201934426 |Validati
Epoch 55: Train acc: 0.6165705515765976, Train loss: 0.015053790884502675 |Validati
Epoch 56: Train acc: 0.6165783027935385, Train loss: 0.014557237698075673 |Validati
Epoch 57: Train acc: 0.6177332341177565, Train loss: 0.015116697116311462 |Validati
```

```
Epoch 58: Train acc: 0.6192989799398505, Train loss: 0.014897766165501838 |Validati
Epoch 59: Train acc: 0.6154853812048492, Train loss: 0.015044878904951648 |Validati
Epoch 60: Train acc: 0.61688060025424, Train loss: 0.014886783624422691 |Validation
Epoch 61: Train acc: 0.61534585929991, Train loss: 0.014854313629690469 |Validation
Epoch 62: Train acc: 0.619748550522432, Train loss: 0.014598669100364316 |Validatio
Epoch 63: Train acc: 0.6189424239605618, Train loss: 0.014660369316559462 |Validati
Epoch 64: Train acc: 0.6174464390909373, Train loss: 0.014773132221307606 |Validati
Epoch 65: Train acc: 0.615066815490032, Train loss: 0.014910019065220175 |Validatio
Epoch 66: Train acc: 0.6193997457600844, Train loss: 0.014454848471186346 |Validati
Epoch 67: Train acc: 0.6192137165534989, Train loss: 0.01426726131328559 |Validatio
Epoch 68: Train acc: 0.6191672092518525, Train loss: 0.014754376303504355 |Validati
Epoch 69: Train acc: 0.6220661643878089, Train loss: 0.014359591876633377 |Validati
Epoch 70: Train acc: 0.6203298917930115, Train loss: 0.01418620972184553 |Validatio
Epoch 71: Train acc: 0.6198570675596069, Train loss: 0.014523119276245347 |Validati
Epoch 72: Train acc: 0.6231590859764983, Train loss: 0.0142335372705323 |Validation
Epoch 73: Train acc: 0.623500139521905, Train loss: 0.014669511306454382 |Validatio
Epoch 74: Train acc: 0.6182448144358664, Train loss: 0.014328515073949737 |Validati
Epoch 75: Train acc: 0.6193299848076148, Train loss: 0.01446969673963308 |Validatio
Epoch 76: Train acc: 0.6194229994109075, Train loss: 0.014143923395923116 |Validati
Epoch 77: Train acc: 0.6163225126344836, Train loss: 0.014544193301671407 |Validati
Epoch 78: Train acc: 0.6227792763463864, Train loss: 0.014375063138903073 |Validati
Epoch 79: Train acc: 0.6191672092518525, Train loss: 0.01371694131964995 |Validatio
Epoch 80: Train acc: 0.6193067311567916, Train loss: 0.014108648177214144 |Validati
Epoch 81: Train acc: 0.6206166868198307, Train loss: 0.014103817581642596 |Validati
Epoch 82: Train acc: 0.6217483644932255, Train loss: 0.0138876151523575 |Validation
Epoch 83: Train acc: 0.6224769788856851, Train loss: 0.01393151059441845 |Validatio
Epoch 84: Train acc: 0.619097448299383, Train loss: 0.013744632983746539 |Validatio
Epoch 85: Train acc: 0.6202678820574831, Train loss: 0.013731642228473598 |Validati
Epoch 86: Train acc: 0.6233373639661427, Train loss: 0.013881543277031077 |Validati
Epoch 87: Train acc: 0.6177177316838743, Train loss: 0.01412470174060824 |Validatio
Epoch 88: Train acc: 0.6152373422627353, Train loss: 0.013760443370778202 |Validati
Epoch 89: Train acc: 0.6243062660837752, Train loss: 0.013670213825424157 |Validati
Epoch 90: Train acc: 0.6220196570861626, Train loss: 0.013775359435905037 |Validati
Epoch 91: Train acc: 0.6221591789911016, Train loss: 0.01371146693903332 |Validatio
Epoch 92: Train acc: 0.6189346727436208, Train loss: 0.013612415709455187 |Validati
Epoch 93: Train acc: 0.6206941989892413, Train loss: 0.013431452332519083 |Validati
Epoch 94: Train acc: 0.621035252534648, Train loss: 0.013729576574405655 |Validatio
Epoch 95: Train acc: 0.6227560226955632, Train loss: 0.01333198331052526 |Validatio
Epoch 96: Train acc: 0.6208414721111214, Train loss: 0.013501073631535595 |Validati
Epoch 97: Train acc: 0.621686354757697, Train loss: 0.013753738114333135 |Validatio
Epoch 98: Train acc: 0.6258797631228102, Train loss: 0.013576779709962596 |Validati
Epoch 99: Train acc: 0.6221901838588658, Train loss: 0.013277314190331492 |Validati
Epoch 100: Train acc: 0.6192059653365578, Train loss: 0.013412867694915 76 |Validati
```




```
#set4
#Based on the first 3 trials, 0.005 is the optimal learning rate.
#My next step is to tune the batch size.
#Base on the training curve in the second trial, there is some noise present
#in the curve, so I decided to increase the batch size to reduce that and see
#if it would further improve the validation accuracy.
#batch_size = 128, num_epochs=100, learning_rate=0.005
net4 = AutoEncoder()
```

```
train(net4, train_data, val_data, batch_size = 128, num_epochs=100, learning_rate=0.005)
```

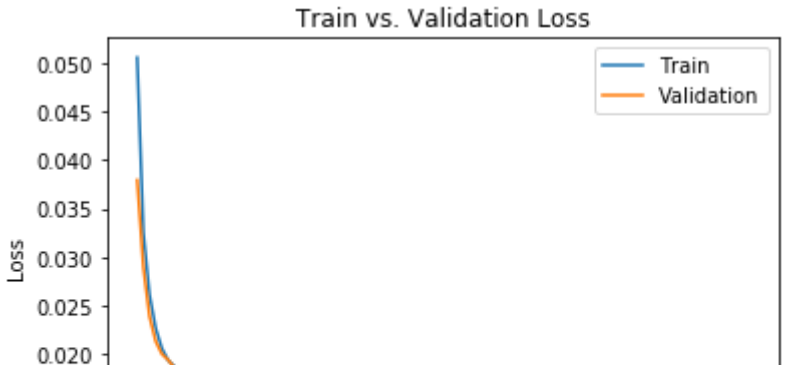
```
Epoch 1: Train acc: 0.571218181254457, Train loss: 0.07338323056076963 |Validation
Epoch 2: Train acc: 0.6027346293368059, Train loss: 0.044427280380789726 |Validatio
Epoch 3: Train acc: 0.6079589495550801, Train loss: 0.03687043297326281 |Validation
Epoch 4: Train acc: 0.6090828760115338, Train loss: 0.0317805431827548 |Validation
Epoch 5: Train acc: 0.6200430967661923, Train loss: 0.028039007880059735 |Validatio
Epoch 6: Train acc: 0.6138498744302856, Train loss: 0.025688477470317765 |Validatio
Epoch 7: Train acc: 0.6115942703004371, Train loss: 0.02445511731673919 |Validation
Epoch 8: Train acc: 0.6111059436331504, Train loss: 0.023265530122444034 |Validatio
Epoch 9: Train acc: 0.6187253898862122, Train loss: 0.022564276376561748 |Validatio
Epoch 10: Train acc: 0.6283368988931263, Train loss: 0.021827889190587615 |Validati
Epoch 11: Train acc: 0.6172139025827055, Train loss: 0.020218673821849126 |Validati
Epoch 12: Train acc: 0.6277090503209004, Train loss: 0.01974998537584075 |Validatio
Epoch 13: Train acc: 0.6308327907481475, Train loss: 0.019370388067770927 |Validati
Epoch 14: Train acc: 0.6349719405946733, Train loss: 0.018576460596661837 |Validati
Epoch 15: Train acc: 0.6272904846060832, Train loss: 0.01772800267540983 |Validatio
Epoch 16: Train acc: 0.6355765355160761, Train loss: 0.01749107303718726 |Validatio
Epoch 17: Train acc: 0.6428781818745543, Train loss: 0.0167768747113379 |Validation
Epoch 18: Train acc: 0.644653210554057, Train loss: 0.016143602202646434 |Validatio
Epoch 19: Train acc: 0.6395451585898986, Train loss: 0.015550313312338577 |Validati
Epoch 20: Train acc: 0.6442191424053576, Train loss: 0.016396377479568833 |Validati
Epoch 21: Train acc: 0.6446764642048801, Train loss: 0.015009824785270862 |Validati
Epoch 22: Train acc: 0.6403512851517689, Train loss: 0.015521384404218267 |Validati
Epoch 23: Train acc: 0.6418550212383344, Train loss: 0.015924560305263315 |Validati
Epoch 24: Train acc: 0.6464980001860292, Train loss: 0.01492187689014134 |Validatio
Epoch 25: Train acc: 0.6469320683347286, Train loss: 0.015340263046146859 |Validati
Epoch 26: Train acc: 0.6509161938424333, Train loss: 0.014130633302210342 |Validati
Epoch 27: Train acc: 0.6488311164852882, Train loss: 0.014646774019292068 |Validati
Epoch 28: Train acc: 0.6495132235761014, Train loss: 0.014127476157487504 |Validati
Epoch 29: Train acc: 0.649962794158683, Train loss: 0.014620951249352879 |Validatio
Epoch 30: Train acc: 0.639862958484482, Train loss: 0.014561182403537844 |Validatio
Epoch 31: Train acc: 0.6526292127864075, Train loss: 0.01499676559719124 |Validatio
Epoch 32: Train acc: 0.6463584782810902, Train loss: 0.014376578140183397 |Validati
Epoch 33: Train acc: 0.652179642203826, Train loss: 0.013829434692438337 |Validatio
Epoch 34: Train acc: 0.6479087216693021, Train loss: 0.013994763239419885 |Validati
Epoch 35: Train acc: 0.6495674820946888, Train loss: 0.014182415688299529 |Validati
Epoch 36: Train acc: 0.6526912225219359, Train loss: 0.013553736268520532 |Validati
Epoch 37: Train acc: 0.6539081635816824, Train loss: 0.013955924942690347 |Validati
Epoch 38: Train acc: 0.6533035686602796, Train loss: 0.013268576937705456 |Validati
Epoch 39: Train acc: 0.6507611695036121, Train loss: 0.012514514108521066 |Validati
Epoch 40: Train acc: 0.6484900629398815, Train loss: 0.013296415760470111 |Validati
Epoch 41: Train acc: 0.6509782035779618, Train loss: 0.013424440700050798 |Validati
Epoch 42: Train acc: 0.6513890180758379, Train loss: 0.013125270799112817 |Validati
Epoch 43: Train acc: 0.6542879732117942, Train loss: 0.01338263242394619 |Validatio
Epoch 44: Train acc: 0.6596750689858307, Train loss: 0.012344406258004407 |Validati
Epoch 45: Train acc: 0.6524199299289989, Train loss: 0.012542866192580689 |Validati
Epoch 46: Train acc: 0.6524044274951167, Train loss: 0.0123448939384183 |Validation
Epoch 47: Train acc: 0.6564118066536446, Train loss: 0.013354157042201786 |Validati
Epoch 48: Train acc: 0.6557839580814188, Train loss: 0.012730305127444722 |Validati
Epoch 49: Train acc: 0.6575124794592752, Train loss: 0.012187958938912266 |Validati
Epoch 50: Train acc: 0.6490714042104611, Train loss: 0.012936555242742457 |Validati
Epoch 51: Train acc: 0.657117167395281, Train loss: 0.012703398514228562 |Validatio
Epoch 52: Train acc: 0.6623569900474374, Train loss: 0.012526104639705625 |Validati
Epoch 53: Train acc: 0.644513688649118, Train loss: 0.01245403891551264 |Validation
Epoch 54: Train acc: 0.6562645335317645, Train loss: 0.012538815050252847 |Validati
Epoch 55: Train acc: 0.6553653923666015, Train loss: 0.012477543166217705 |Validati
Epoch 56: Train acc: 0.661008278299693, Train loss: 0.012711149474073733 |Validatio
Epoch 57: Train acc: 0.6583961181905559, Train loss: 0.011977565677149133 |Validati
```

```
Epoch 58: Train acc: 0.6583573621058506, Train loss: 0.012650871078395062 |Validati
Epoch 59: Train acc: 0.657101664961399, Train loss: 0.012620796995525737 |Validatio
Epoch 60: Train acc: 0.659837844541593, Train loss: 0.011957900404619673 |Validatio
Epoch 61: Train acc: 0.6565203236908195, Train loss: 0.01220455777920073 |Validatio
Epoch 62: Train acc: 0.6550398412550771, Train loss: 0.012107001728422585 |Validati
Epoch 63: Train acc: 0.651652559451834, Train loss: 0.012269188727562627 |Validatio
Epoch 64: Train acc: 0.654644529191083, Train loss: 0.011836009205407685 |Validatio
Epoch 65: Train acc: 0.6497690137351564, Train loss: 0.011884361236644466 |Validati
Epoch 66: Train acc: 0.6599773664465322, Train loss: 0.012136881411563428 |Validati
Epoch 67: Train acc: 0.6581945865500883, Train loss: 0.012121498191152654 |Validati
Epoch 68: Train acc: 0.6588456887731374, Train loss: 0.01240139075144682 |Validatio
Epoch 69: Train acc: 0.6587759278206679, Train loss: 0.012279868455758939 |Validati
Epoch 70: Train acc: 0.6588999472917249, Train loss: 0.011355801582491646 |Validati
Epoch 71: Train acc: 0.6637909651815335, Train loss: 0.011904930626596547 |Validati
Epoch 72: Train acc: 0.6616283756549778, Train loss: 0.011670395249633916 |Validati
Epoch 73: Train acc: 0.6547530462282578, Train loss: 0.011927781044505537 |Validati
Epoch 74: Train acc: 0.6614113415806282, Train loss: 0.011594821809835378 |Validati
Epoch 75: Train acc: 0.6655117353424488, Train loss: 0.01221637769846157 |Validatio
Epoch 76: Train acc: 0.6587294205190215, Train loss: 0.01135477826132306 |Validatio
Epoch 77: Train acc: 0.6607679905745202, Train loss: 0.011455475025632907 |Validati
Epoch 78: Train acc: 0.666441881375376, Train loss: 0.01175037697456511 |Validation
Epoch 79: Train acc: 0.6668526958732521, Train loss: 0.011804134925893908 |Validati
Epoch 80: Train acc: 0.666705422751372, Train loss: 0.011399465261049391 |Validatio
Epoch 81: Train acc: 0.6552181192447214, Train loss: 0.011274963467647987 |Validati
```

```
#set5
#batch_size = 32, num_epochs=100, learning_rate=0.005
#It turned out that a larger batch size does not help improve the validation
#accuarcy for this model.
#So I will train this model with a smaller batch size to see what would
#happen. A small batch size might help the algorithm jump out of a local minimum.
net5 = AutoEncoder()
train(net5, train_data, val_data, batch_size = 32, num_epochs=100, learning_rate=0.005)
```

```
Epoch 1: Train acc: 0.5995256255232071, Train loss: 0.050526375230956114 |Validatio
Epoch 2: Train acc: 0.6076876569621431, Train loss: 0.03255679774918549 |Validation
Epoch 3: Train acc: 0.6011766347316528, Train loss: 0.02613212776902531 |Validation
Epoch 4: Train acc: 0.6005720398102502, Train loss: 0.022681761344539978 |Validatio
Epoch 5: Train acc: 0.6056568381235854, Train loss: 0.020620981340817662 |Validatio
Epoch 6: Train acc: 0.6146017424735684, Train loss: 0.019420012101320373 |Validatio
Epoch 7: Train acc: 0.6133227916782935, Train loss: 0.01876107280605495 |Validation
Epoch 8: Train acc: 0.6182758193036306, Train loss: 0.01829446557981317 |Validation
Epoch 9: Train acc: 0.6144932254363935, Train loss: 0.017804460004082926 |Validatio
Epoch 10: Train acc: 0.6202058723219546, Train loss: 0.0173864910528729 |Validation
Epoch 11: Train acc: 0.6039748240473755, Train loss: 0.016924346893488075 |Validati
Epoch 12: Train acc: 0.6260270362446905, Train loss: 0.01645228912128091 |Validatio
Epoch 13: Train acc: 0.6305925030229746, Train loss: 0.015478987089452511 |Validati
Epoch 14: Train acc: 0.6339022726568071, Train loss: 0.014943053717919005 |Validati
Epoch 15: Train acc: 0.6375685982699284, Train loss: 0.01494716878057391 |Validatio
Epoch 16: Train acc: 0.6444051716119431, Train loss: 0.014076422000860995 |Validati
Epoch 17: Train acc: 0.6188494093572691, Train loss: 0.014195718210733807 |Validati
Epoch 18: Train acc: 0.6380104176355688, Train loss: 0.014207591413564626 |Validati
Epoch 19: Train acc: 0.6460406783865067, Train loss: 0.013993440127316197 |Validati
Epoch 20: Train acc: 0.6409481288562304, Train loss: 0.013382578979612194 |Validati
Epoch 21: Train acc: 0.6493737016711624, Train loss: 0.013202345493135951 |Validati
Epoch 22: Train acc: 0.6523346665426472, Train loss: 0.013045407915515603 |Validati
Epoch 23: Train acc: 0.6428084209220848, Train loss: 0.0128189238641339 |Validation
Epoch 24: Train acc: 0.6450175177502868, Train loss: 0.012997536298089904 |Validati
Epoch 25: Train acc: 0.6554041484513069, Train loss: 0.012531242591822297 |Validati
Epoch 26: Train acc: 0.6524896908814684, Train loss: 0.012630064787767228 |Validati
Epoch 27: Train acc: 0.6554351533190711, Train loss: 0.012563697005957476 |Validati
Epoch 28: Train acc: 0.6556134313087155, Train loss: 0.012614307129253922 |Validati
Epoch 29: Train acc: 0.6509084426254922, Train loss: 0.012465567582347873 |Validati
Epoch 30: Train acc: 0.6581093231637367, Train loss: 0.01201721846674835 |Validatio
Epoch 31: Train acc: 0.6474048925681332, Train loss: 0.012439368947525509 |Validati
Epoch 32: Train acc: 0.6531562955383995, Train loss: 0.01189148739615034 |Validatio
Epoch 33: Train acc: 0.6568226211515208, Train loss: 0.01196865173308955 |Validatio
Epoch 34: Train acc: 0.6557762068644777, Train loss: 0.011712282694393903 |Validati
Epoch 35: Train acc: 0.6583651133227917, Train loss: 0.011595620419636058 |Validati
Epoch 36: Train acc: 0.6574659721576287, Train loss: 0.011906347996652281 |Validati
Epoch 37: Train acc: 0.6582565962856168, Train loss: 0.011736651475768582 |Validati
Epoch 38: Train acc: 0.6508231792391406, Train loss: 0.011466066257049707 |Validati
Epoch 39: Train acc: 0.6473351316156637, Train loss: 0.01143508720893546 |Validatio
Epoch 40: Train acc: 0.6570706600936347, Train loss: 0.011760316018245206 |Validati
Epoch 41: Train acc: 0.6572024307816328, Train loss: 0.011512122291952394 |Validati
Epoch 42: Train acc: 0.6610547856013395, Train loss: 0.0116750454153156 |Validation
Epoch 43: Train acc: 0.6620624438036772, Train loss: 0.011396232976126373 |Validati
Epoch 44: Train acc: 0.6601556444361765, Train loss: 0.011308425130707855 |Validati
Epoch 45: Train acc: 0.6617213902582706, Train loss: 0.01129470518970352 |Validatio
Epoch 46: Train acc: 0.6562567823148234, Train loss: 0.011554146843471764 |Validati
Epoch 47: Train acc: 0.665922549840325, Train loss: 0.011410352314539653 |Validatio
Epoch 48: Train acc: 0.658923200942548, Train loss: 0.01102758476952745 |Validation
Epoch 49: Train acc: 0.6596518153350076, Train loss: 0.011243925927244294 |Validati
Epoch 50: Train acc: 0.6679533686788826, Train loss: 0.01102456835603031 |Validatio
Epoch 51: Train acc: 0.6641552723777633, Train loss: 0.011113583378443894 |Validati
Epoch 52: Train acc: 0.6617291414752116, Train loss: 0.011439679483980095 |Validati
Epoch 53: Train acc: 0.6624887607354355, Train loss: 0.011134064884529272 |Validati
Epoch 54: Train acc: 0.6662248473010263, Train loss: 0.011000290370984225 |Validati
Epoch 55: Train acc: 0.6549545778687254, Train loss: 0.011066773239387354 |Validati
Epoch 56: Train acc: 0.6517145691873624, Train loss: 0.011370129221870143 |Validati
Epoch 57: Train acc: 0.6614113415806282, Train loss: 0.010982240704747494 |Validati
```

```
Epoch 58: Train acc: 0.660101385917589, Train loss: 0.0113988653798255 |Validation
Epoch 59: Train acc: 0.6588534399900784, Train loss: 0.011008421798621947 |Validati
Epoch 60: Train acc: 0.6690230366167488, Train loss: 0.0111344547362283 |Validation
Epoch 61: Train acc: 0.6655194865593899, Train loss: 0.011294595165790747 |Validati
Epoch 62: Train acc: 0.6612873221095712, Train loss: 0.01103460286339257 |Validatio
Epoch 63: Train acc: 0.6647908721669302, Train loss: 0.011038663482828443 |Validati
Epoch 64: Train acc: 0.6664806374600812, Train loss: 0.01120752211482752 |Validatio
Epoch 65: Train acc: 0.6630313459213096, Train loss: 0.010987829679964176 |Validati
Epoch 66: Train acc: 0.6661395839146746, Train loss: 0.010955025474513172 |Validati
Epoch 67: Train acc: 0.6701624655070846, Train loss: 0.010917681767924 |Validation
Epoch 68: Train acc: 0.6635816823241248, Train loss: 0.010716640328090372 |Validati
Epoch 69: Train acc: 0.6643645552351719, Train loss: 0.011024119638022967 |Validati
Epoch 70: Train acc: 0.6644343161876415, Train loss: 0.011374830441560508 |Validati
Epoch 71: Train acc: 0.6635506774563606, Train loss: 0.010711533439115599 |Validati
Epoch 72: Train acc: 0.6614500976653335, Train loss: 0.011105627569902157 |Validati
Epoch 73: Train acc: 0.6673177688897157, Train loss: 0.011198039945849728 |Validati
Epoch 74: Train acc: 0.6645893405264627, Train loss: 0.011165032067981969 |Validati
Epoch 75: Train acc: 0.6709530896350727, Train loss: 0.010973994008617453 |Validati
Epoch 76: Train acc: 0.6585743961802003, Train loss: 0.011228992802595409 |Validati
Epoch 77: Train acc: 0.6663721204229064, Train loss: 0.011088581272772336 |Validati
Epoch 78: Train acc: 0.6655349889932719, Train loss: 0.011110237683169544 |Validati
Epoch 79: Train acc: 0.6637599603137693, Train loss: 0.010532215951873999 |Validati
Epoch 80: Train acc: 0.6659535547080891, Train loss: 0.010851474418207848 |Validati
Epoch 81: Train acc: 0.6654807304746845, Train loss: 0.010795116750939217 |Validati
Epoch 82: Train acc: 0.6603339224258209, Train loss: 0.011182644756142205 |Validati
Epoch 83: Train acc: 0.6646901063466965, Train loss: 0.010729656340983985 |Validati
Epoch 84: Train acc: 0.6680541344991163, Train loss: 0.01090000747527007 |Validatio
Epoch 85: Train acc: 0.6646048429603447, Train loss: 0.010935279055307287 |Validati
Epoch 86: Train acc: 0.6644033113198772, Train loss: 0.011006164739401789 |Validati
Epoch 87: Train acc: 0.6647831209499891, Train loss: 0.010858171336751963 |Validati
Epoch 88: Train acc: 0.6614733513161566, Train loss: 0.01039490801243422 |Validatio
Epoch 89: Train acc: 0.6687982513254581, Train loss: 0.01136381007970721 |Validatio
Epoch 90: Train acc: 0.6575512355439804, Train loss: 0.010684635560166845 |Validati
Epoch 91: Train acc: 0.6666666666666666, Train loss: 0.010935235414113518 |Validati
Epoch 92: Train acc: 0.666720925185254, Train loss: 0.01111414647346551 |Validation
Epoch 93: Train acc: 0.6727358695315164, Train loss: 0.010585619200615579 |Validati
Epoch 94: Train acc: 0.6679223638111184, Train loss: 0.010819839251953886 |Validati
Epoch 95: Train acc: 0.6653412085697454, Train loss: 0.010948987001375783 |Validati
Epoch 96: Train acc: 0.6689687780981615, Train loss: 0.010708410026078733 |Validati
Epoch 97: Train acc: 0.6596750689858307, Train loss: 0.010462169210292077 |Validati
Epoch 98: Train acc: 0.6686432269866369, Train loss: 0.010829517920466884 |Validati
Epoch 99: Train acc: 0.66515517936316, Train loss: 0.010664906046475239 |Validation
Epoch 100: Train acc: 0.6629848386196633, Train loss: 0.010706447147683883 |Validat
```




```
#By either increasing or decreasing the batch size, the model's performance
#is not improved.
#So I will keep the batch size at 64.
```

```
#It is time to pick the optimal num_epoch to prevent overfitting.
#I picked the model that has the highest validation accuracy and applied early
#stopping to that model.
best = AutoEncoder()
model_path = get_model_name(best.name, batch_size=64, learning_rate=0.005,
                            epoch=77)#actual epoch=90, num_epoch starts from 0
state = torch.load(model_path)
best.load_state_dict(state)
```

```
IncompatibleKeys(missing_keys=[], unexpected_keys=[])
```

# Part 4. Testing [12 pt]

## Part (a) [2 pt]

Compute and report the test accuracy.

```
Final Training Accuracy: 0.6629848386196633
```

```
test_loader = torch.utils.data.DataLoader(test_data, batch_size=64, num_workers=1, shuffle
test_acc = get_accuracy(best, test_loader)
print("The test accuracy is "+str(test_acc))
```

```
The test accuracy is 0.7015335648148148
```

**The test accuracy (70.1%) is almost the same as the validation accuracy (70.3%) but a slightly lower, which is reasonable. Because the hyperparameters are tuned based on the validation set.**

## Part (b) [4 pt]

Based on the test accuracy alone, it is difficult to assess whether our model is actually performing well. We don't know whether a high accuracy is due to the simplicity of the problem, or if a poor accuracy is a result of the inherent difficulty of the problem.

It is therefore very important to be able to compare our model to at least one alternative. In particular, we consider a simple **baseline** model that is not very computationally expensive. Our neural network should at least outperform this baseline model. If our network is not much better than the baseline, then it is not doing well.

For our data imputation problem, consider the following baseline model: to predict a missing feature, the baseline model will look at the **most common value** of the feature in the training set.

For example, if the feature "marriage" is missing, then this model's prediction will be the most common value for "marriage" in the training set, which happens to be "Married-civ-spouse".

What would be the test accuracy of this baseline model?

**Do not actually implement this baseline model. You should be able to compute the test accuracy by reasoning about how the basline model behaves.**

**Whenever asked to make a prediction, the baseline model will pick the most common value of the feature. Based on that, to get the test accuracy of the model, we just need to check the average number of the most common value for all features and the test accuracy should be close to that.**

**I will check the actual test accuracy by iterating through all the test data and comparing its each category value to the most common value of that feature. If the value matches, I increment the correct result by 1. The final test accuracy will be (#correct )/(#elements in the catcols in the test data).**

```
baseline_result = {}
for feature in catcols:
  start_index = cat_index[feature]
  stop_index = cat_index[feature] + len(cat_values[feature])
  records = train_data[:, start_index:stop_index]
  records_sum = np.sum(records, axis = 0)
  idx_max = np.argmax(records_sum)
  baseline_result[feature] = cat_values[feature][idx_max]
print(baseline_result)

#get accuaracy of the algorithm
total = 0
acc = 0
for col in catcols:
  for record in test_data: # test_data is the same in part1 g
    acc += int(baseline_result[col] == get_feature(record, col))
    total += 1
test_acc = acc / total
print("The total test accuracy is: {}".format(test_acc))
```

```
{'work': 'Private', 'marriage': 'Married-civ-spouse', 'occupation': 'Prof-specialty
The total test accuracy is: 0.4568504050925926
```

### Part (c) [1 pt]

How does your test accuracy from part (a) compared to your basline test accuracy in part (b)?

**My test accuracy is part(a) is 25%higher than the baseline test accuracy in part(b). This high percentage difference is a evident indication that my autoencoder performs well.**

### Part (d) [1 pt]

Look at the first item in your test data. Do you think it is reasonable for a human to be able to guess this person's education level based on their other features? Explain.

```
first_features = {}
for col in catcols:
  first_features[col] = get_feature(test_data[0], col)

print(first_features)
```

```
{'work': 'Private', 'marriage': 'Divorced', 'occupation': 'Prof-specialty', 'edu':
```

**Note: I only displayed the first data's features in catcols, but other features not included in catcols are also helpful.**

**Yes. I think it is reasonable to guess this person's education level based on his other features. Some of the closely related features to education are occupation, yredu and capgain. A person with higher capgain is more likely to have higher education level as he is more familiar and is aware of the importance of investing his wealth. Another clue for his education level is occupation. His occupation is Prof-specialty. People with high education level is more likely to have such occupation. Reasonable guesses based on these two clues are "Bachlors", "Masters" or "PHD".**

**The most direct is the feature eduyr, we can basically tell a person's education level directly based on his eduyr unless this person has grade skipping or has taken any gap years.**

### Part (e) [2 pt]

What is your model's prediction of this person's education level, given their other features?

**I first created a copy of the first test data and zero out all the entries that belongs to "edu" category.**

**Then, I fed in the zeroed_feature data to my trained model and used get_feature function to obtain the prediction.**

```
sample = torch.tensor(test_data[0])
start_index = cat_index["edu"]
stop_index = cat_index["edu"] + len(cat_values["edu"])
sample[start_index:stop_index] = 0
out = best(sample).detach().numpy()
pred = get_feature(out, "edu")
print(pred)
```

```
Bachelors
```

**Given other features (work, marriage, occupation, relationship, sex) , my prediction is the same as the ground truth.**

### Part (f) [2 pt]

What is the baseline model's prediction of this person's education level?

```
baseline_result["edu"]
```

```
'HS-grad'
```

**Same as part b, the base line model made predictions based on the most common value of the feature.**

**The most common value in "edu" category is "HS-grad", which does not match the actual value. This is actually a good indication that our model performs well on this particular problem.**