Coursework 2 - Part 1 - Report

Part 1: Dataset analysis and define a suitable setup

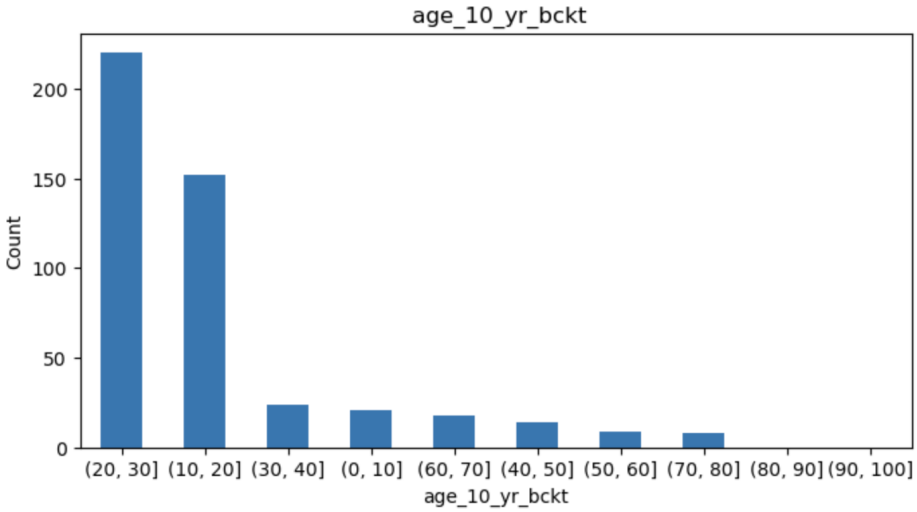
Data Analysis

General analysis of the data

- There is no missing data in the table: There are no nulls in the meta dataframe table, and the numerical columns do not contain any zeros that might be suspected as missing data.
- The data contain 466 participants.
- The data contain 8 numerical columns and 7 categorical columns. Each of those categories should be analysed separately.

Categorical columns

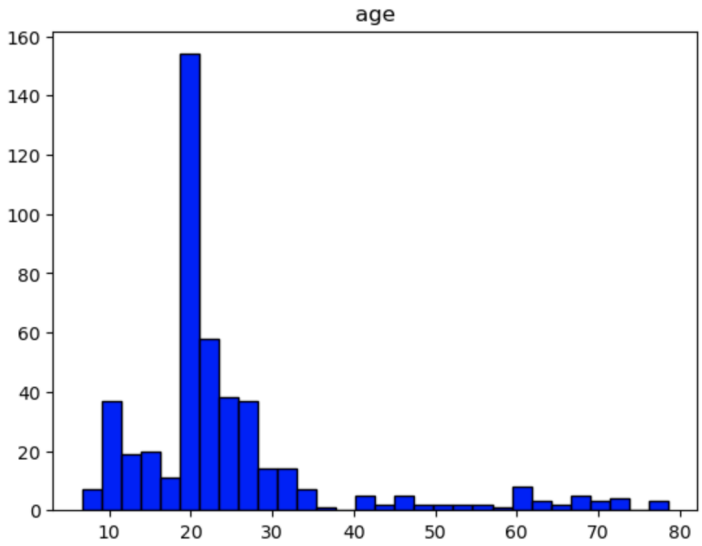
- The number of male and female participants is quite similar, with **225 (48.3%) female participants and 241 (51.7%) male participants.**
- All participants are from the “control” group, as they represent healthy patients. Therefore the “diagnosis” column can be ignored.
- The vast majority of the participants’ ages belong to the buckets “10-20” or “20-30”.



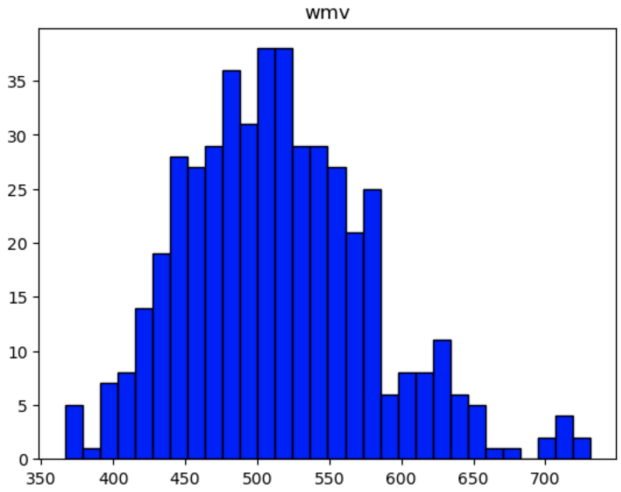
- In fact, participants aged 10-30 account for over 79% of the participants in the data.

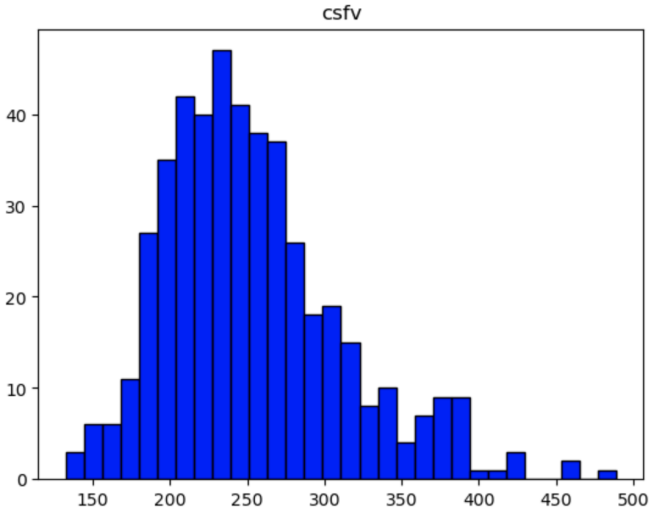
Numerical columns

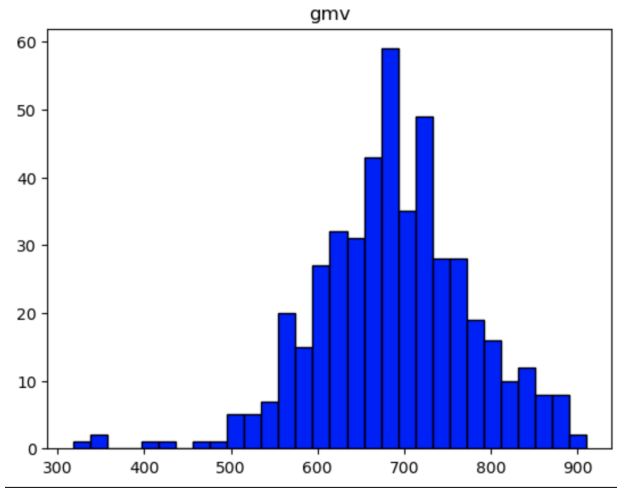
- To be more specific about the imbalance of the data, there are 157 participants between the ages of 19 and 21, which account to over ⅓ of the dataset. This may create a problem of overfitting of the model to younger ages. On the other hand, there are very few participants in the ages of 37 and older: only 49 participants which account for only 10.5% of the data.



- WMV, GMV and CSVF distributions looks pretty normal, except for a few participants with extreme values that will require further investigation.



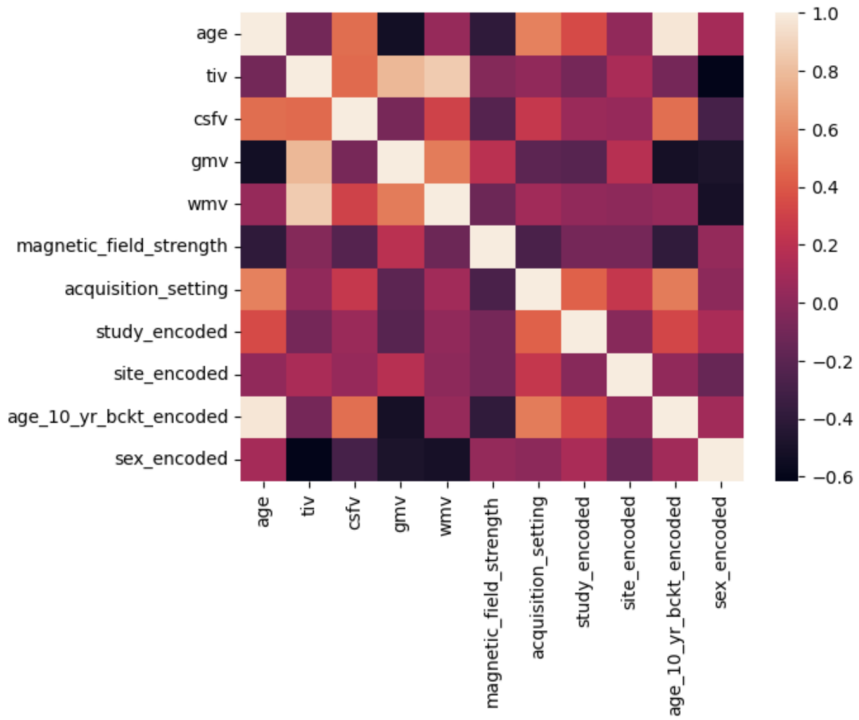




-

Correlation between features

For deepening the understanding the characteristics of the different volumes, it is worth analysing the correlation they have with the other features in the dataset. Attached bellow is a heatmap of the features

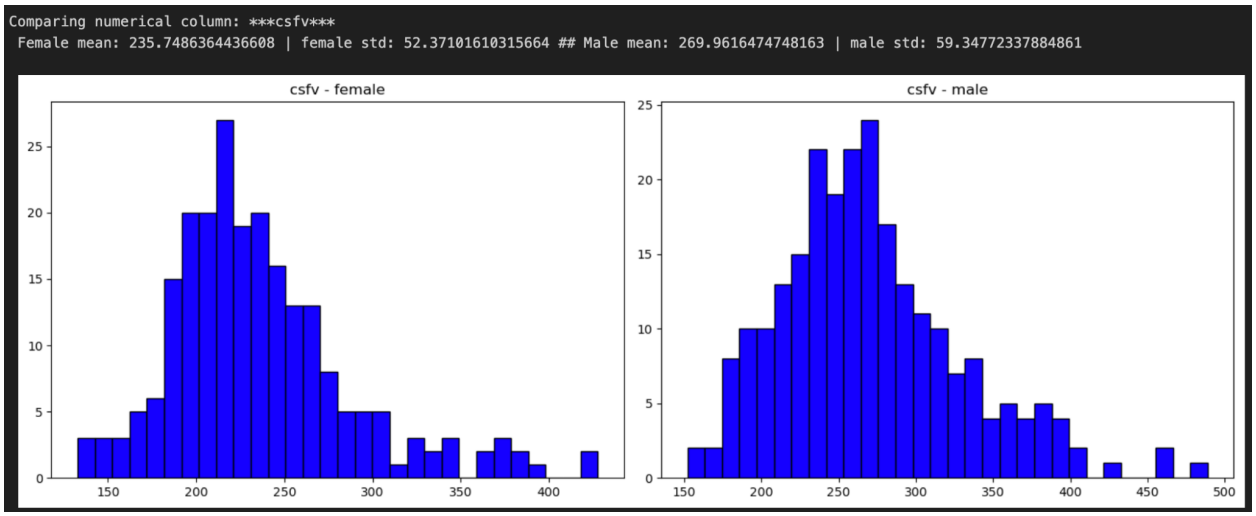


Positive indication: csfv has a positive correlation with the age, and gmv has negative correlation with the age. This is a good sign, as the more correlation between those features to the age, the more simple it is to predict age based on those features.

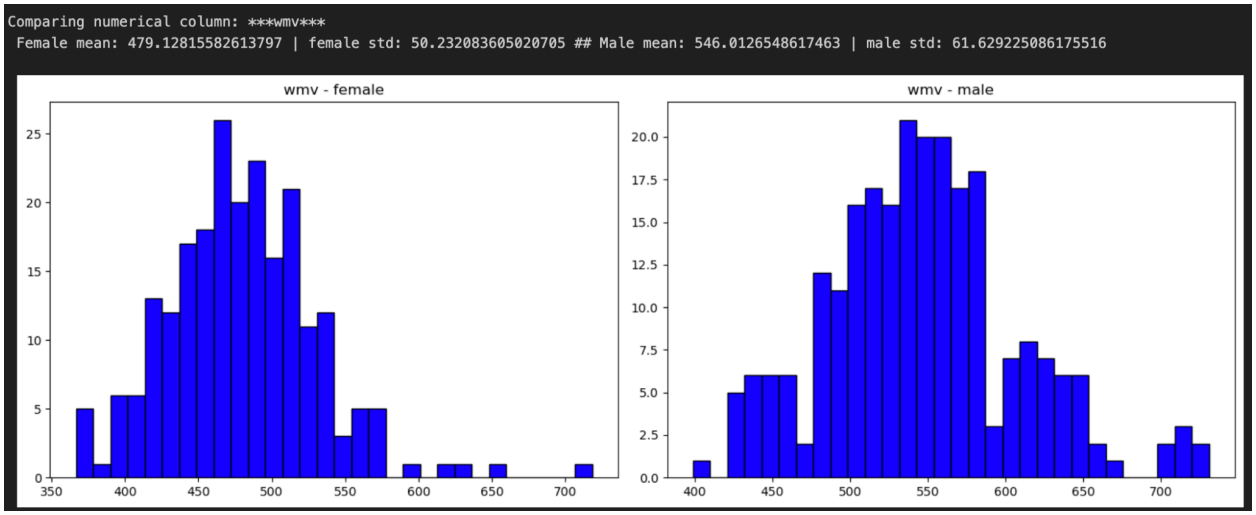
Important discovery: **Both csfv, gmv and wmv have a relatively strong negative correlated with the encoded sex of the patient**. This might be problematic when trying to predict the age of the patient using those 3 values, as the age is independent from the sex of the patient.

To further understand the problem and the potential solutions, here is a **comparative analysis of gmv, wmv and csfv between female and male patients:**

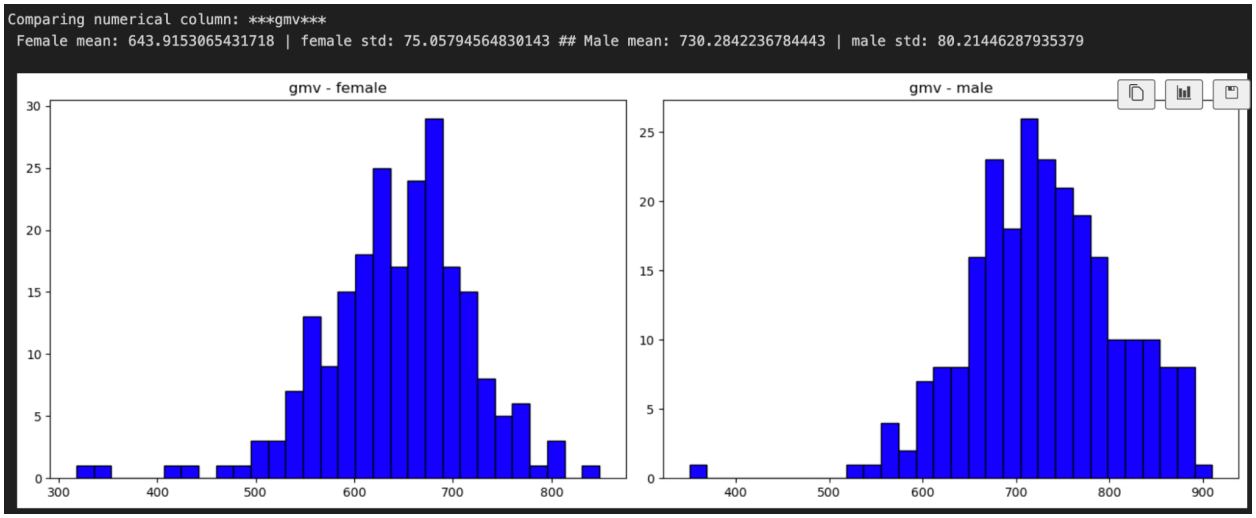
CSFV - female mean=235.75 | male mean=269.96



WMV - female mean=479.13 | male mean=546.01



GMV - female mean=643.91 | male mean=730.28



Note: In both wmv and gmv, **the mean volume of female patients is lower in more than one std than the mean of male patients.**

The data analysis have shown two potential issues that may have impact on the performance of the model, and therefore may influence the decisions about its design.

- Some extreme values for wmv, csfv and gmv
- Unwanted correlation between the sex of the patient to the various brain volumes.

These may influence the design of the pipeline, as I wish to ensure that the sex of the patient will not have any impact on the error the model make when predicting their age, and clearly the sex of the patient should not have any correlation with their age (at least for the ranges of the experiment, which are between 0 and 80).

Defining a suitable setup:

Train/validation/test split

I chose to use 80% of the training data for training the model, and 20% for validation of the model. The training set is used for the model to learn patterns and adjust its parameters to perform predictions. This is the only data that the model is exposed to during the training loop, and therefore the only data uses for adjusting the model's parameters during the backward pass. The validation set is used to evaluate the model's ability to generalize what it learnt, by assessing the model's performance on unseen data. Validation data is used for not only assessing the model's performance, but also for model selection, hyperparameters tuning, and detecting whether the model exhibits overfitting to the training set.

The coursework also contain test data that is set aside during the entire process of developing and selecting a model. This test set will be used eventually for final evaluation of the model's performance of unseen data. Since the validation set is used for model selection and hyperparams tuning, the test set is used for an unbiased evaluation of the model's performance

on data that was not taken into account at any point during the model development. Therefore, the test set is used to calculate the model's precision.

Stratification

Stratification is used for ensuring that certain groups have equal representation in training, validation and test sets. It is especially useful in cases of imbalance data, like the data in this case, to avoid a situation where training, test or validation sets have a skewed distribution.

I decided to stratify on "sex" variable, and on "age_10_yr_bckt_bg_70":

- "sex" variable: Since females and males have different physical characteristics, it makes sense to ensure both train and validation set have equal representation of each.
- "Age_10_yr_bckt_bg_70" variable: The training set contains much more young patients than older patients. As shown in the data analysis section: over 79% of the participants are between the ages 10 to 30, but only 10.5% of the participants are 37 years or older. This makes it particularly important to validate that all age groups are represented in both training and validation sets.

Test Metric

I have decided to use mean absolute error as my main test metric. Although quadratic loss could help in prioritizing small errors over large errors, I found that the absolute error makes it particularly easy to understand the magnitude of the error and interpret it in the context of the real world problem: To understand exactly by how many years the model was wrong.

Part 2: Baseline model definition

Justification for your volume prediction model structure/design

The baseline model for volume prediction is composed of one convolutional layer with kernel of 5*5*5, followed by leaky relu activation function and average pool of cubes of 2*2*2. It is then followed by a linear layer to take all the outputs, and calculate the 3 required outputs of csfv, gmv and wmv.

The main consideration behind those design choices was to find the most simple and lightweight model possible, and make sure it is able to learn something, so that it can be used as a baseline for improvement.

1. 3D Convolutional layer with kernel of 5*5*5: The convolution layer can be useful to detect patterns. To be more specific, it is focused on detecting local patterns, which is particularly relevant in this case, since the model needs to detect volume of grey matter, white matter and cerebrospinal fluid, each located in a different location within the brain.

The convolutional layer can help focus on more local patterns that are relevant for each of those matters.

2. 3d Average pooling: The pooling reduces the dimension of the input for the next layer, which will naturally affect the number of parameters the model contains and the optimizer has to optimize. This is particularly important in the case of this coursework, as the training data only contain 365 samples, with 96*96*96 (884,736) pixels each. As the pooling occurs *after* the convolution, and as this is an average pooling, it is guaranteed that each of the "pooled" pixels will contain information on its local neighboring pixels.
3. Linear layer to reduce the result of convolution into the final prediction: Since this is a base version of the model, the simplicity of the model is more important than the optimization of it at this point. A single linear layer is easy to understand and debug, and therefore could be a good choice for the base model.

I decided to use MSE as my loss function for two reasons:

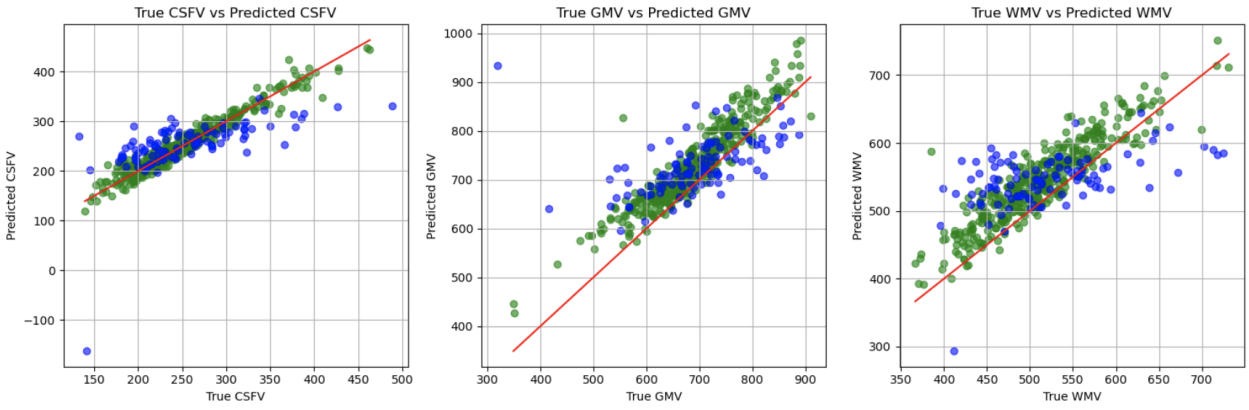
Firstly, it is twice derivable and smooth, which may make it easier to work with in the optimization of a complicated convolutional neural network. Secondly, in the context of predicting volumes in the brain, the main advantage of Mean Absolute Error becomes less relevant: While when measuring error in age prediction it is extremely intuitive measure the error in the absolute number of years, brain volumes are different: They are much less intuitive, and it makes much more sense to have higher loss on predictions with a large error, while punishing less on predictions that are closer to the truth.

Discussion of results from volume prediction model

Results

Volume regression results

- Train loss (normalized MSE) 0.084129
- Test loss (normalized MSE) 1.041788
- Test MAE (Denormalized) 52.271903
- Predictions plot:

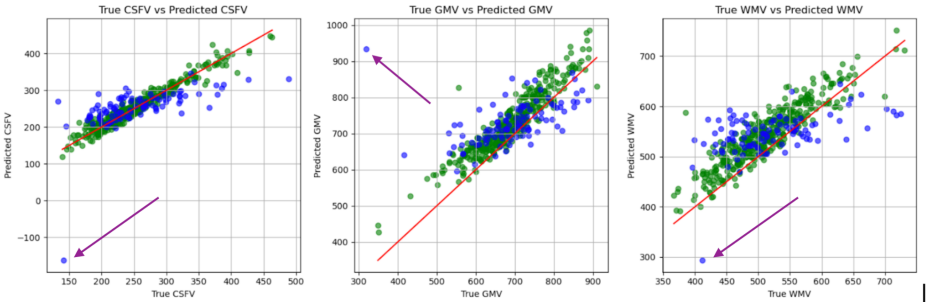


Discussion

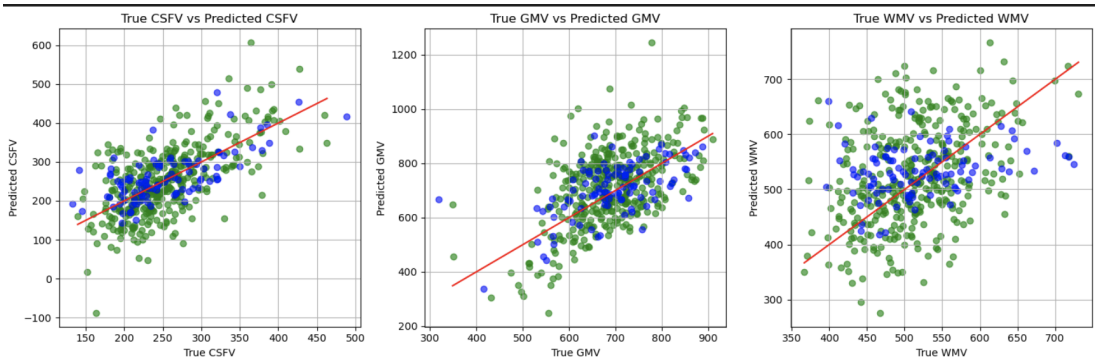
Are the results reasonable?

For checking whether the results are a good baseline to start with, I chose to compare the volume prediction models to two simple methods:

1. **Predicting the average**
    a. I implemented a short loop to get the validation error for a model that simply returns the average volumes, regardless of the MRI input. The results were:
    `Test MSE loss: 1.244120 | Test MAE loss: 60.502063`
    This means that the baseline model I implemented, with the scores
    `Test loss (normalized MSE) 1.041788 | Test MAE (Denormalized) 52.271903`
    Is doing better than guessing the average. Moreover, note that my base model had one extreme error in each of the different volumes on **one specific example**, which significantly increases the average error of the model, although the extreme error is just for one patient in each volume (which might even turn out to be the same one).



2. **Applying linear regression directly to the MRI images**
    a. The result of a basic linear regression applied directly on the images is as follow:
    `Train loss: 1.892824 | Test loss: 1.039815 | Test MAE: 10.485066.`
    As a reminder, my base model's results are:
    `Train loss: 0.084129 | Test loss: 1.041788 | Test MAE: 52.271903`
    So, the linear model's results *on test set* are slightly better than my base model's results on the test set, which required some more investigation regarding whether the Linear regression could perform as a base model for this task. Eventually I have decided it is not good enough, for the following reasons:
        i. The linear regression model's results on the training set are quite bad compared to my base model `(0.084129)`, which suggest that the model wasn't able to learn well enough the features and the patterns of the image. Meaning, the model lacks the expressivity needed to capture the relevant features of the MRIs.
        Attached is a plot of the linear regression's performance



These plots demonstrates the fact that the model wasn't able to learn well enough from the training set.

ii. When trying to continue and optimize the linear regression model, the result became much worse, which was another indicator of the fact that the model lacks the ability to learn the patters from iterating over the data

<u>Further discussion of the results:</u>

Overall, the results of my base volume prediction model looks reasonable for a baseline, with a few points to note:

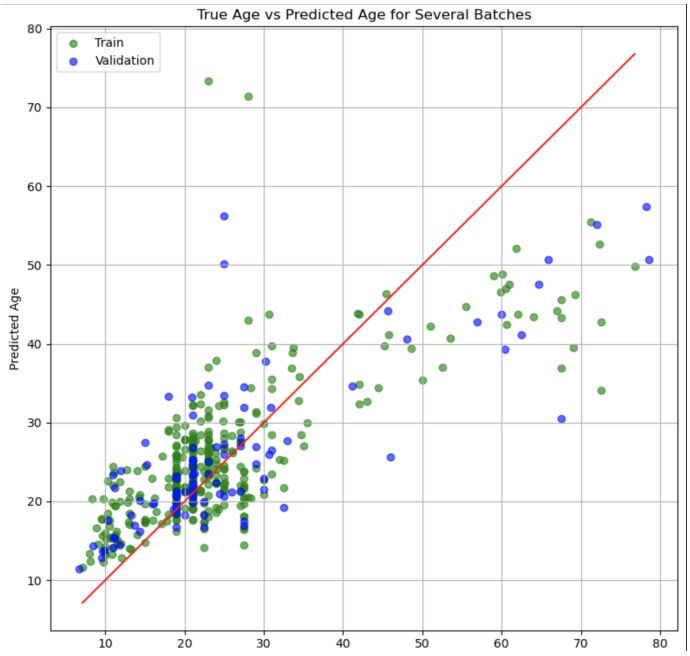
1. One point to note, is that for each of the tasks (csfv, wmv, gmv), there is one prediction that is out of bounds, and seems completely off relatively to the rest of the predictions. This might require looking into removing outliers
2. The model seems to struggle in predicting high csfv, which could be explained by the fact that high csfv is correlated with older patients, and the model was trained on relatively low number of those. The model also exhibits some overfitting and seems to struggle the most in predicting the WMV.

MLP regression model

(after changing number of epochs from 30 to 10 to avoid overfitting) -

Results

- Train loss (normalized data): 0.537097
- Test loss (normalized data): 0.577078
- Test Mean absolute error (denormalized data): 6.916039 (meaning, on average the model predicts on validation set age which is 6.9 years far from the real age of the participant)
- Predictions plot:

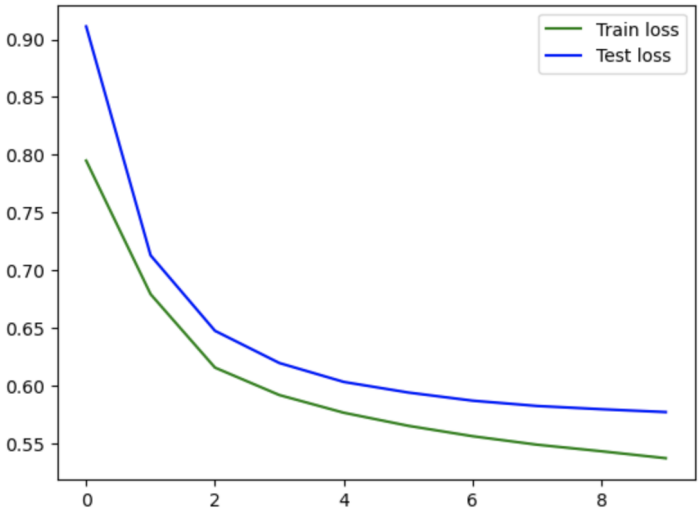


Discussion:

The model produces predictions with mean error of 6.83 years.

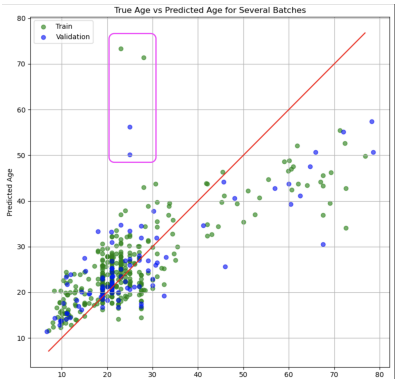
The model doesn't exhibit any signs of over fitting. This could be seen both in the plot of the predictions of train and validations above, where the predictions on validation set (blue) seems to be as far from the truth (red line) as the predictions on training set (green).

Another indication for the fact that the model doesn't exhibit overfitting is the plot of the train loss verses validation loss during the training, where I changed the number of iterations to 10 in order to avoid overfitting, and the losses seems quite close to each other:



Two points worth noting:

1. The model predicts 4 young participants to be 50 or older (2 in train set and 2 in validation set).

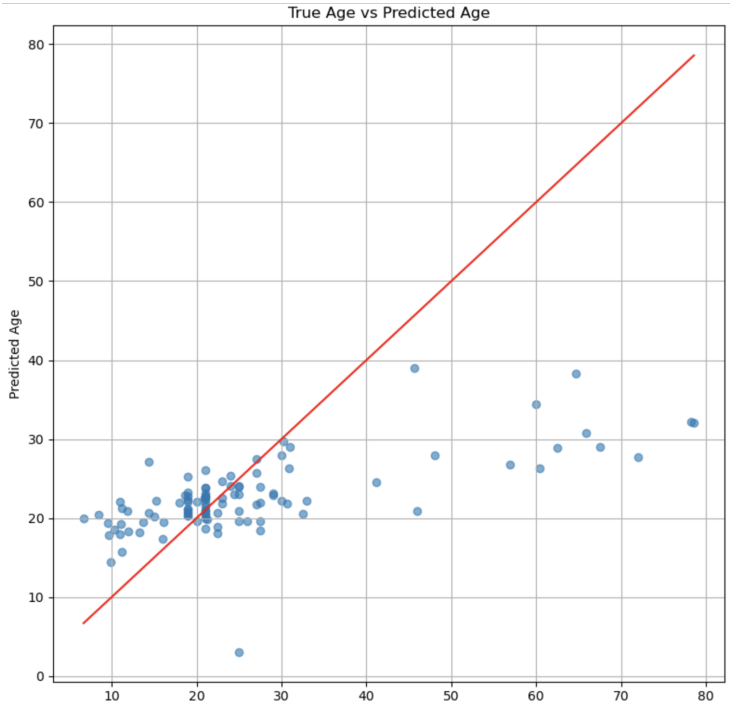


2. The model provides relatively good predictions for participants of ages 18-35 (except for the 4 errors mentioned before), with errors for both sides (either predicting a patient is older or younger than the ground truth). However, for older participants, the model predict them to be younger than their age, with the highest prediction for older participants being below the age of 60. On the other hand, the model predict teenage participants to be older than they are. In my opinion, these errors could be a result of the age distributions in the data, where over ⅓ of the participants are between the ages of 19 and 21.

End-to-end combined model

Results

1. Test loss: 1.089519
2. Mean Absolute Loss: 8.364075
3. Predictions plot (on validation set)



Discussion:

The model is successful in predicting younger ages to younger participants and older age to older participants. However, the combined model shows problem with the range of the predicted ages: Except for one prediction (that looks like an outlier), the model only predicts ages between 15 and 40, while successfully assigning the older ages within this range to the older participants, and the younger ages to the younger participants. Interestingly, the model predicts the correct ages only to some participants in the ages of 19-30. For younger participants, the model predict them to be older. For older participants, the model predicts them to be younger.

In my opinion, this could be a result of the fact that the data is imbalance and contains many participants of young ages (with over ⅓ of the training set between the ages of 19 and 21). That way, the model is always more safe to "guess" a prediction arround those ages, because the chances are it will be closer to the real prediction, due to the demographic of the dataset.

Are the results sufficient to be a reasonable baseline?

In order to answer this question, I calculated the performance of a model that always predicts the average age of all participants. This "model"'s performance on validation set is:
`MSE Test loss: 1.474024, MAE Test loss: 10.409928.`

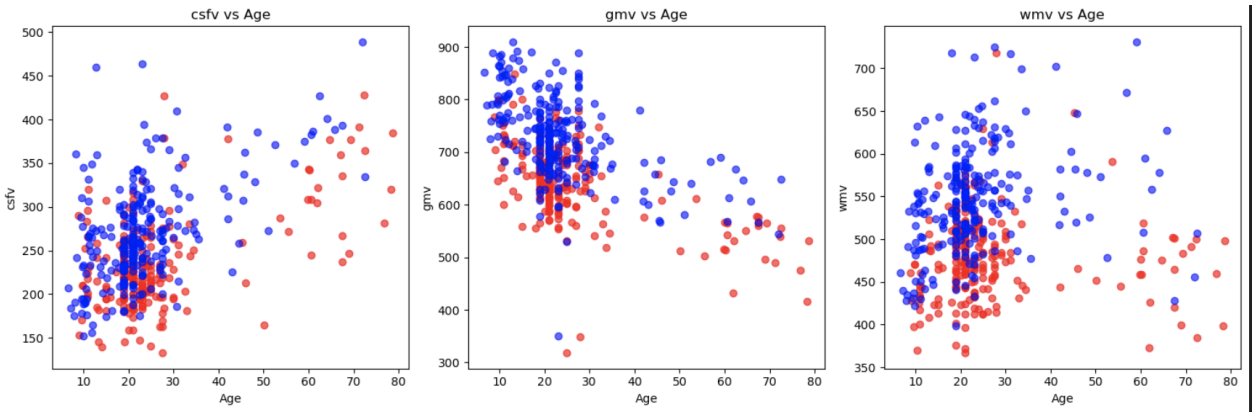
When compared to my combined model (`MSE Test loss: 1.089519, MAE Test loss: 8.364075`), it shows that my base model adds value, it is better than a simple "guess", and can be used as a baseline for the future improvements.

Part 3: Improving upon the Baseline

Hypothesis 1 - Using proportion of volume, instead of the absolute volume

Motivation
The motivation for this hypothesis came from the initial analysis of the data, where I found that there is, on average, a difference between the different volumes of the brain maters for females and males. This was initially indicated in the correlation matrix, that showed correlation between both csfv, gmv and wmv to the encoded sex of the patient (presented in section 1 of this report).

Further to the distribution and the mean shown in part 1, I also plotted the correlation between brain volumes and age, splitted between males and females for better understanding of the phenomena:

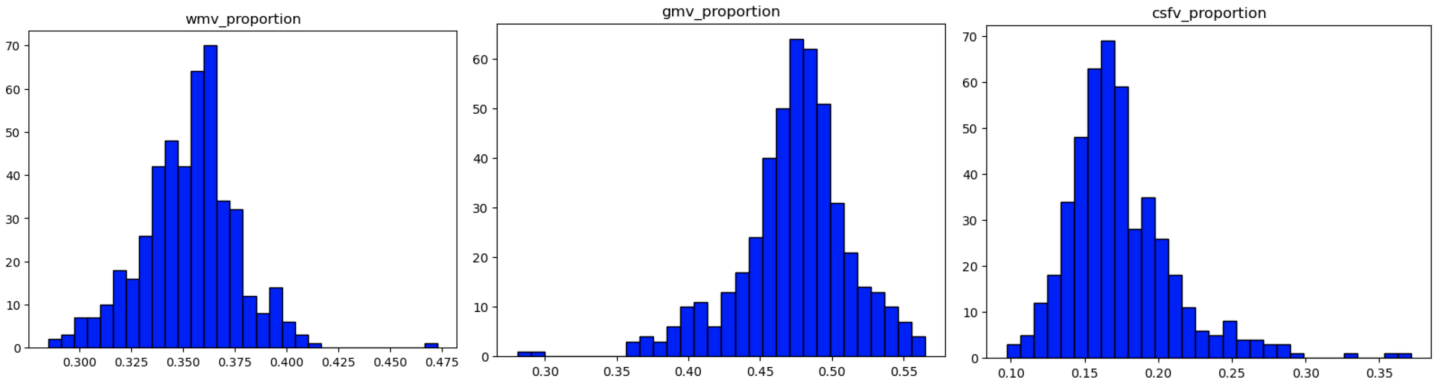


This plot shows that while the trends are similar between the sexes, the general volumes of maters in female brains looks lower.

Hypothesis

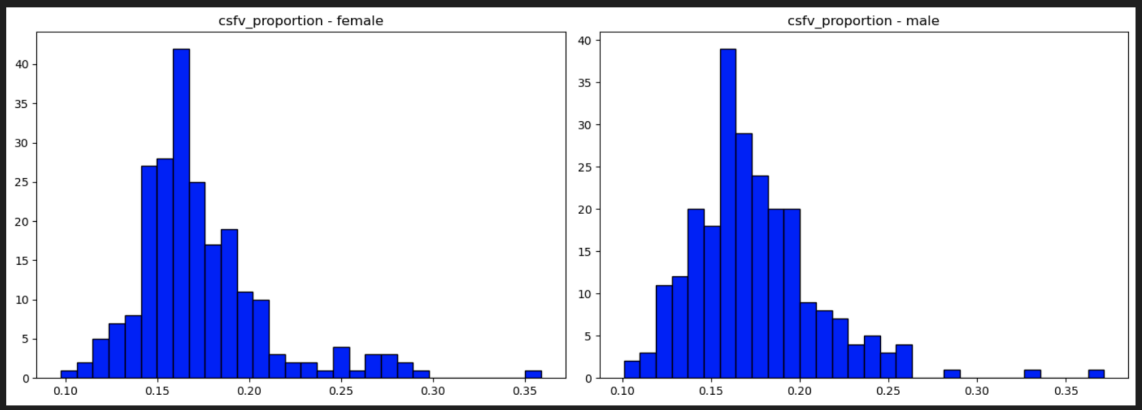
To reduce the influence that the sex have on the prediction, I decided to use the proportion of each mater's volume out of the sum of the volumes of this patient, hoping that it will neutralize the impact of the actual size of the volume.

I have created 3 new columns called csfv_proportion, gmv_proportion, wmv_proportion. Where csfv_proportion = csfv / (csfv + wmv + gmv), and the others are calculated similarly. Attached are the plots describing the distribution of each of those new columns:

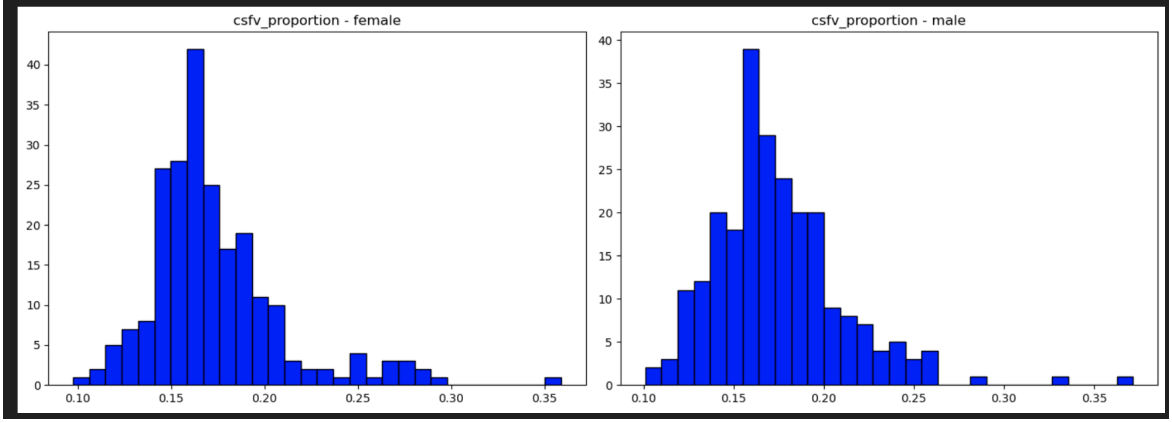


To understand whether these column may solve the problem, attached are the results of a comparison of this feature's distribution between females and males:

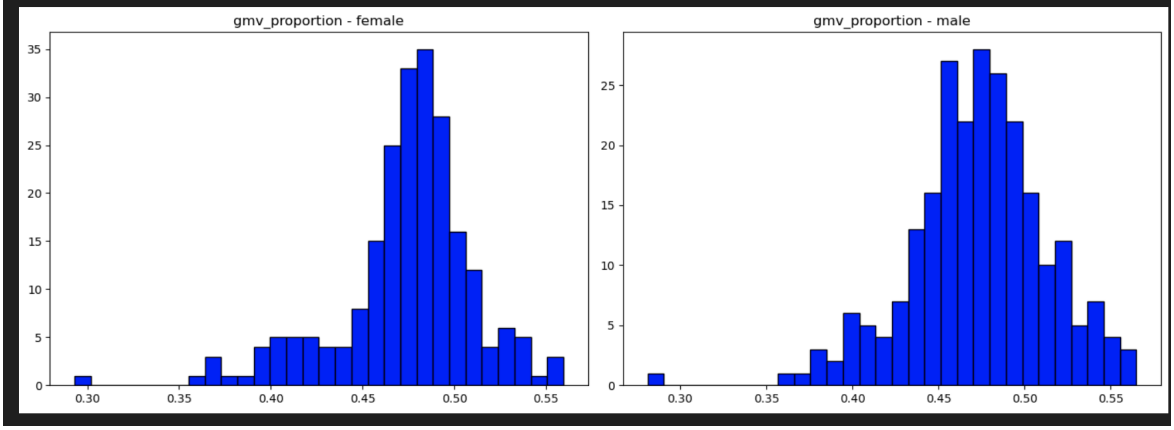
```
Comparing numerical column: ***csfv_proportion***
 Female mean: 0.17 | female std: 0.04 ## Male mean: 0.17 | male std: 0.04
```



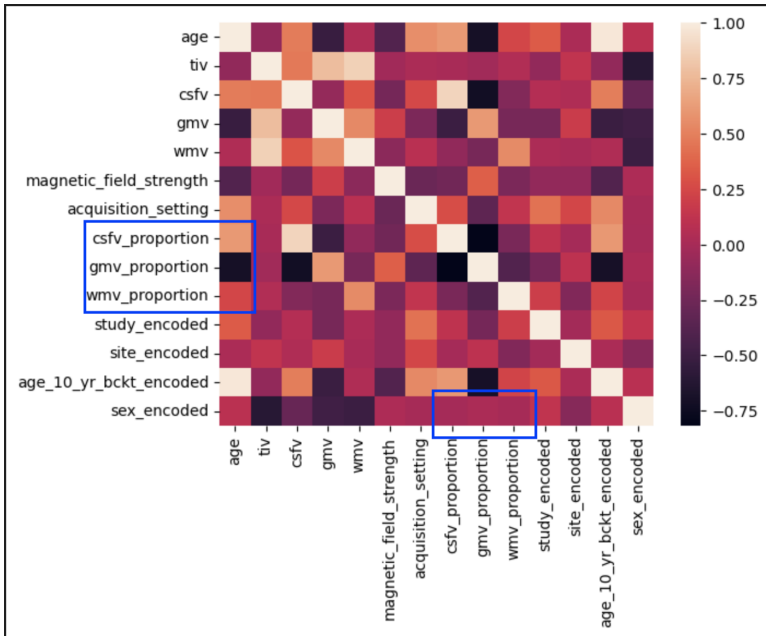

```
Comparing numerical column: ***csfv_proportion***
 Female mean: 0.174 | female std: 0.04 ## Male mean: 0.175 | male std: 0.04
```




```
Comparing numerical column: ***gmv_proportion***
 Female mean: 0.474 | female std: 0.04 ## Male mean: 0.473 | male std: 0.04
```

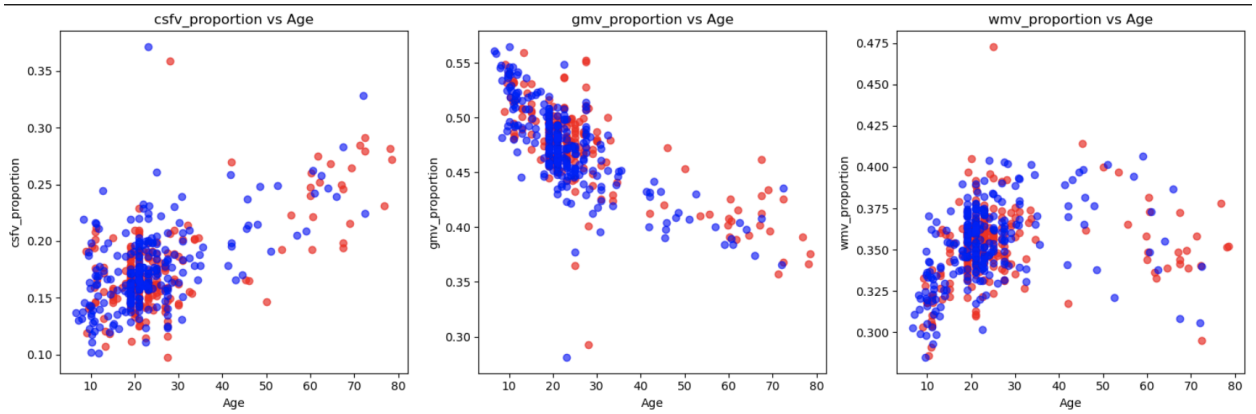


In addition, attached a correlation matrix that includes the new proportioned features:



This matrix shows two good results. Firstly, there is no correlation between the proportion features and the encoded sex. Secondly, we can see that the proportion features have higher correlation (either negative or positive) with the age than their respective un-proportioned volumes.

One final evidence of the logic behind this change is the plot of correlation between the proportion of the different volumes to the age, separating to women and men:



This plot shows that not only do women and men have similar values of this proportioned volume, but also the correlation between this variable and the age seems a bit stronger in those plots than in the absolute volume plot. This is inline with the correlation matrix above, that shows the correlation is stronger.

Experiment:

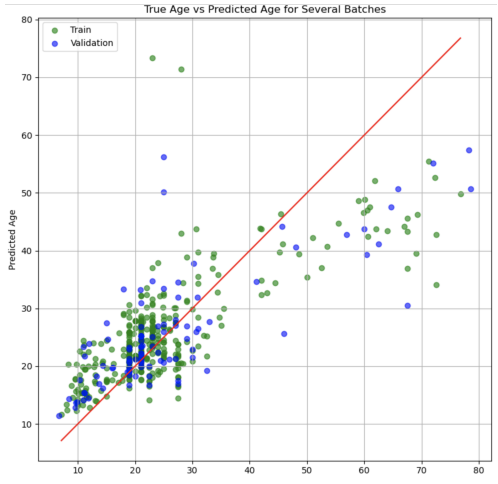
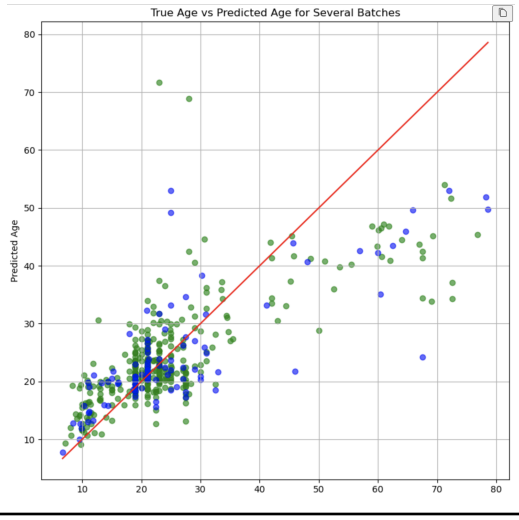
My hypothesis was that using proportioned volumes instead of the absolute volumes will improve the performance of both models. In order to test it, I created dataloaders that uses this feature instead of the absolute volumes, and trained models with exactly the same architecture as the baseline models. I also re-calculated the norm of each of those new “proportion” features,

as the mean and standart deviation of those features is different than those of the absolute volumes.

Required evidence to confirm/disprove the hypothesis:
In order to confirm my hypothesis, I would expect to see that the models which were trained with the proportion features will perform better than the base model on validation set.

Results:

MLP - Age regression

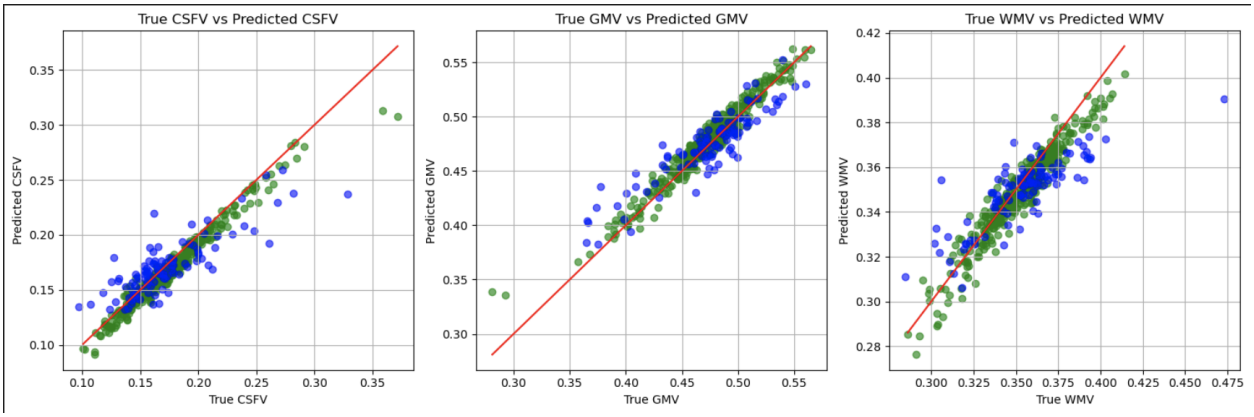
Base	**H1**	**Comparison**
0.537097	0.537016	**MSE on Train**
0.577078	0.611833	**MSE on Validation**
6.916039	6.859985	**MAE on Validation**
		**Plot**

Unfortunately, the performance of the age regression model doesn't confirm the hypothesis for this model. However, while the results are very similar, the proportion MLP shows a slightly better performance on the youngest participants, especially those bellow 10, on which the base model predict they are slightly older.

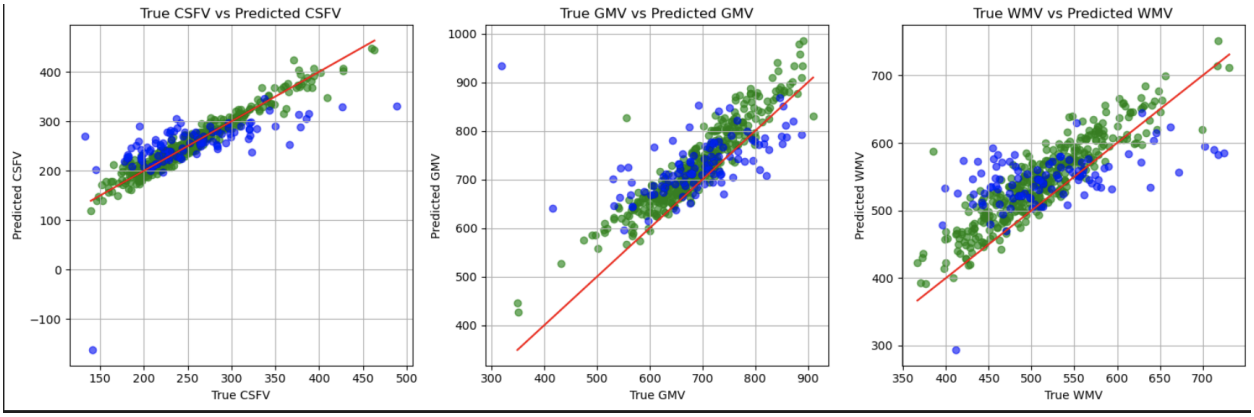
Volume prediction
Note- the error here is a bit hard to compare: Denormalized absolute error is completely irrelevant, due to the face that proportion scale is 0 to 1 while volume scale is in hundreds. The normalized MSE is also different, but we will use it here. The real test will be in the end to end model, where both models are tested with years.

Base	**H1**	**Comparison**
0.084129	0.105431	**MSE on Train**
1.041788,	0.403257	**MSE on Validation**
Not Relevant, as the scale is completely different between the volumes and proportions	Not Relevant, as the scale is completely different between the volumes and proportions	**MAE on Validation - Not relevant**

Plot - H1



Plot - Base

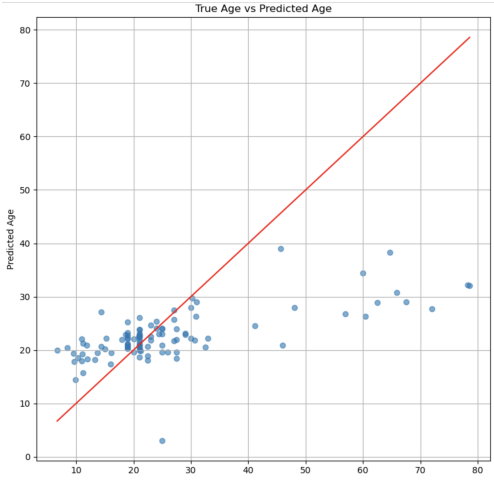
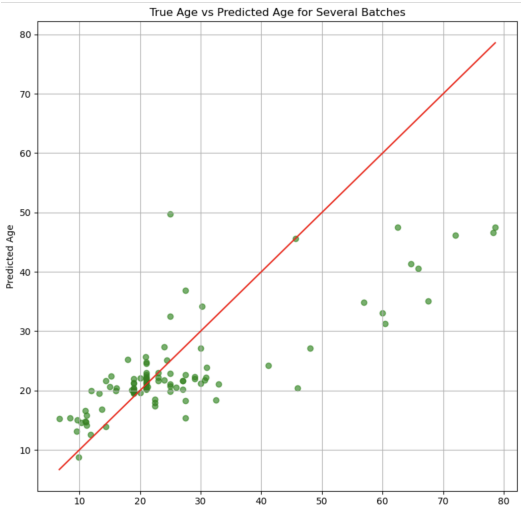


Discussion:

We can see that the proportioned method provides better handling of some of the outliers, and that it enables slightly better results in the prediction of all volumes. Specifically the WMV. However, as the scales are so different between the two problems, it is necessary to compare the combined model

Combined model

Base	**H1**	**Comparison**

1.089519	0.685904	**MSE on Validation**
8.364075	7.012982	**MAE on Validation**
		**Plot**

The combined model performs better than the base: It predicts better especially on younger and older participants: It increases the range, and provides predictions from below 10 years old to almost 50.

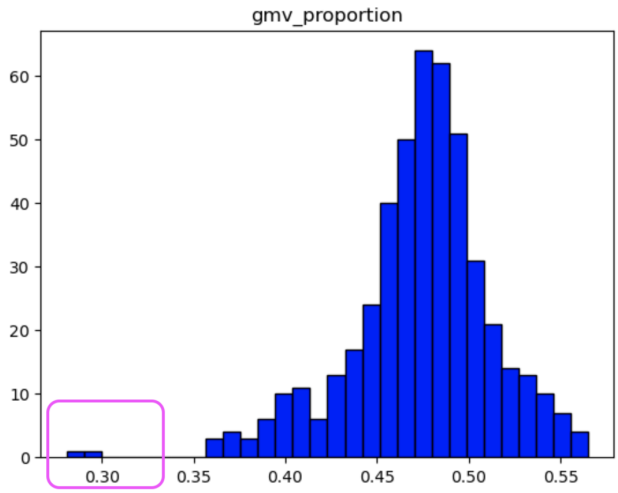
Hypothesis 2 - Remove outlier from proportioned data

Motivation:

The motivation for removing outlier comes from both the data analysis of H1, and from the model results:

In the data analysis of the newly added columns, there were signs for outliers that needed further inspection:

GMV proportion



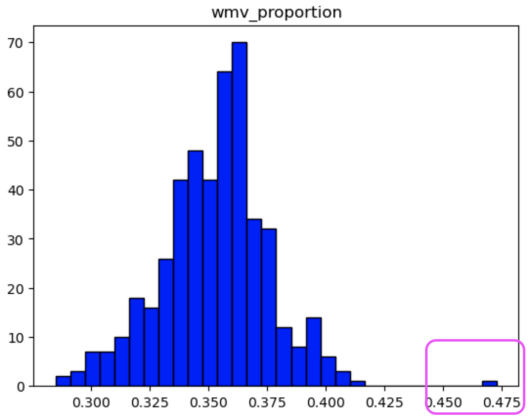
In order to decide whether those should be considered outliers, I checked their rows in the data, and compared with the age/proportioned volume correlation plot above:

```
num of outliers with low gmv proportion: 2
['145200850196' '167010474107']
```

	participant_id	age	sex	gmv_proportion	wmv_proportion	csfv_proportion
161	145200850196	23.0	male	0.280989	0.347341	0.371670
227	167010474107	28.0	female	0.292982	0.348172	0.358846

Recall that low gmv is correlated with older ages, and those participants are young. Which increases the suspicion they should be treated as outliers.

<u>WMV proportion:</u>

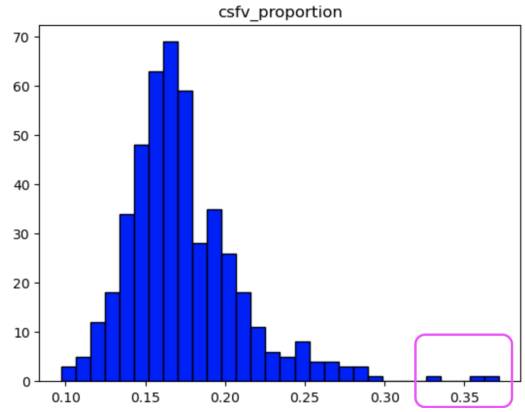


Similarly, I checked for more information about this outlier:

	participant_id	age	sex	gmv_proportion	wmv_proportion	csfv_proportion
432	222716353013	25.0	female	0.365151	0.472886	0.161963

Although this participant is arround the age where wmv peaks, her value is still way far from the rest of the participants which made me decide to treat her as an outlier

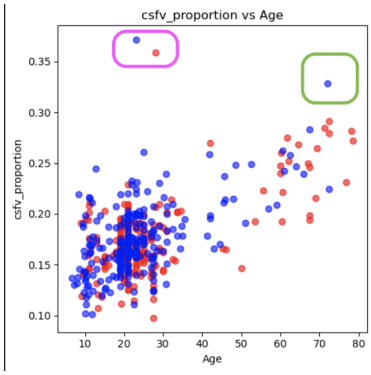
<u>CSFV proportion:</u>



Similarly, I checked for more information about these outliers:

	participant_id	age	sex	gmv_proportion	wmv_proportion	csfv_proportion
144	142102321393	72.0	male	0.365801	0.305933	0.328265
161	145200850196	23.0	male	0.280989	0.347341	0.371670
227	167010474107	28.0	female	0.292982	0.348172	0.358846

Comparing the findings with the correlation plot bellow, I have decided to only treat the younger participants with higher csfv as outliers, and keep the datapoint of the 72 years old male with a higher csfv, as this didn't seem like an outlier from the plot bellow.



Note that the two younger patients with high csfv are already marked as outliers due to their high proportion of gmv, which is a good sign.

Experiment:

Based on the analysis above, I have decided to remove the 3 participants: ['145200850196' '167010474107' '222716353013'].

There were two design decisions about the experiment that I would like to detail:

1. In order for the evaluation and comparison with baseline model and h1 model to be valid, I have chosen to only remove the outliers that are part of the training set, while keeping any outlier patient that is part of validation set for evaluating the model. This way I can be sure that both model are evaluated exactly on the same set of data.
2. However, as I believe those 3 patients are outliers, I have decided to calculate the volume norm with the clean dataset, to reflect the distribution that I believe to be right, without any outliers to skew it.

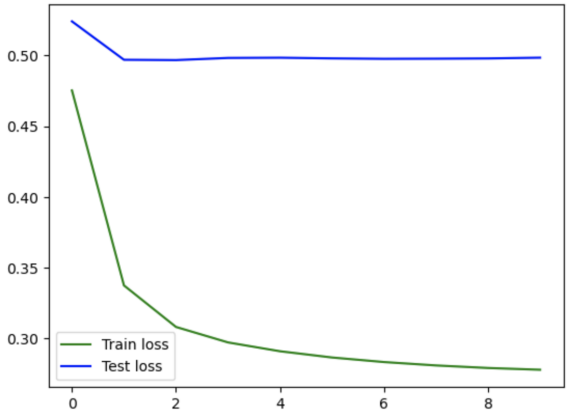
Required evidence to confirm/disprove the hypothesis:

In order to confirm the evidence, I would expect to see that the model performs better than the H1 model on validation set.

Results

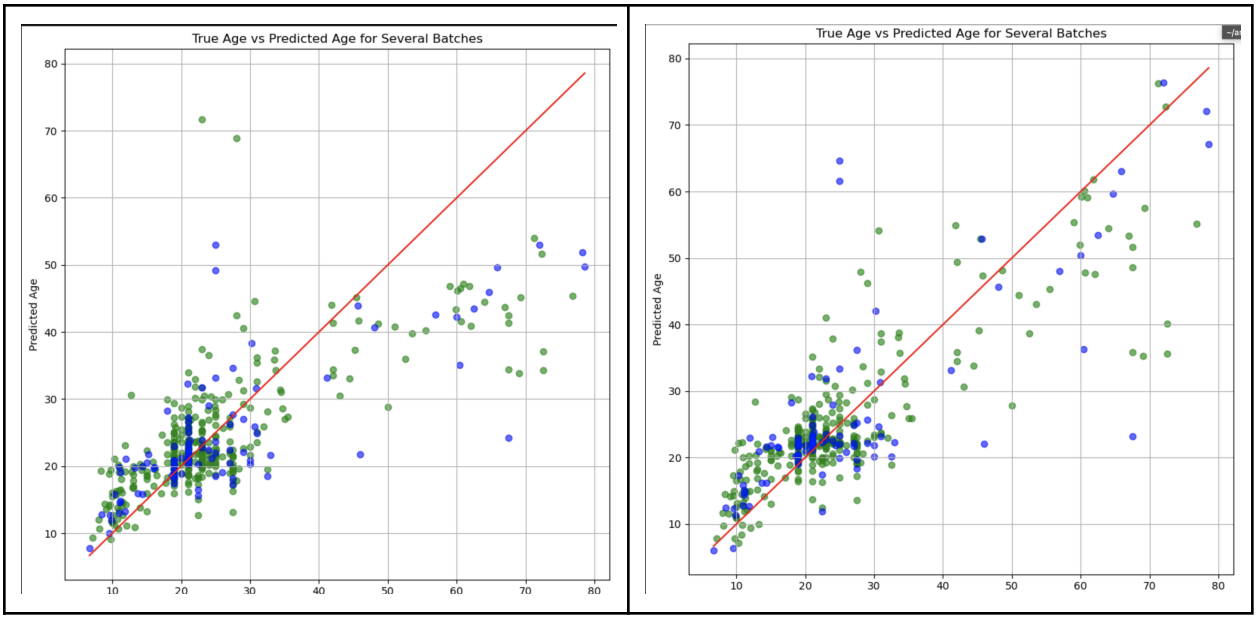
MLP Age regression

Base	**H1**	**H2**	**Comparison**
0.537097	0.537016	0.277760	**MSE on Train**
0.577078	0.611833	0.498469	**MSE on Test**
6.916039	6.859985	5.839305	**MAE on Test**



Overall, the results seems better than both base and H1 models. However, the results seems to look way better on training set, while the error on validation set is has not reduced proportionally. This may be a result of the outlier I chose to keep in the validation set for the comparison to be correct.
Lets compare the plot to the one of H1:

MLP H1	MLP H2

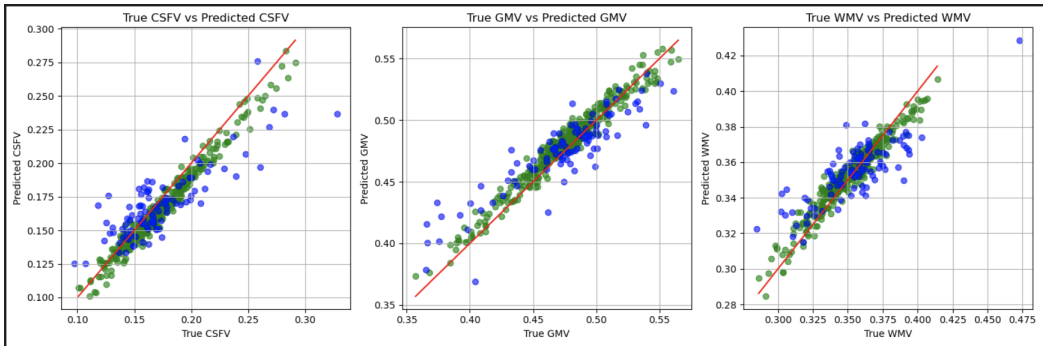
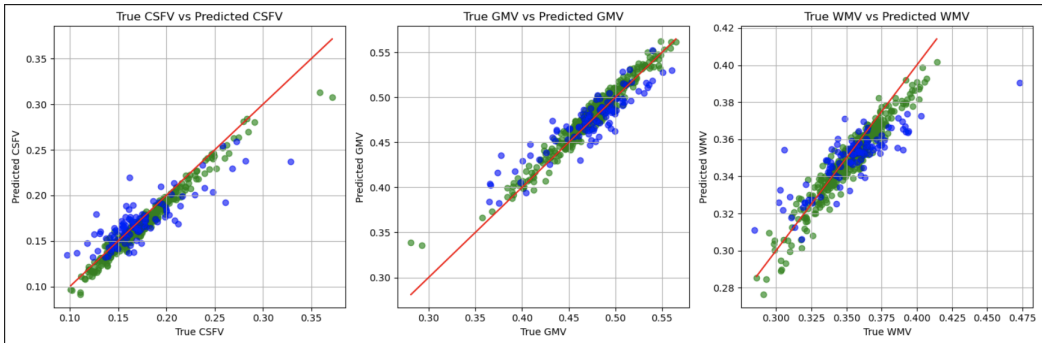


The plot shows that the regression can better predict ages of older patients, while performing slightly better on the yonder patients as well. However, we can see that the unsuccessful predictions of H1 (predicting 25 old patient to be 50 or 53), have became worse in H2, and now the model predicts them to be even older.

Volume Prediction

Base	**H1**	**H2**	**Comparison**
0.084129	0.105431	0.072922	**MSE on Train**
1.041788,	0.403257	0.438615	**MSE on Validation**
Not Relevant	0.014638	0.015278	**MAE on Validation**

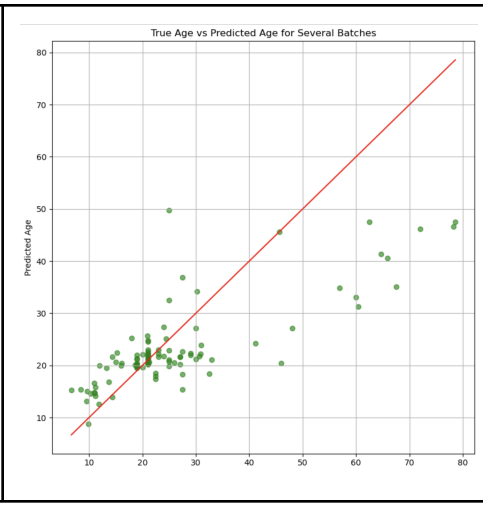
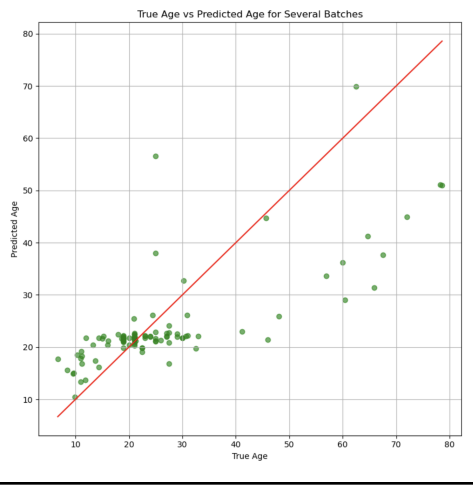
As in the age prediction, the volume prediction shows similar results, where the model's ability to learn on train set improves, while the model's error on validation is slightly increased.

H2 vol plot	
H1 vol plot	

The results look quite similar (note that removing two extreme points from training set changed the scale a little). One thing worth noting is that it provided a better prediction for the patient with the highest WMV proportion: while H1 predicted the patient with 0.475 wmv proportion to have 0.39, the H2 model predicted it to be over 0.42.

Combined Model:

Base	**H1**	**H2**	**Comparison**
1.089519	0.685904	0.682367	**MSE on Validation**
8.364075	7.012982	7.082492	**MAE on Validation**

			**Plot**

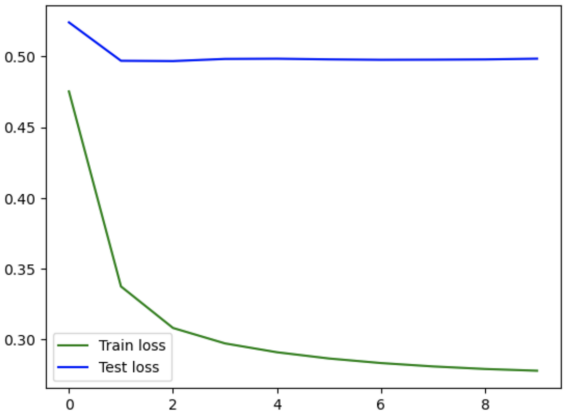
Overall, the results of the combined model for the H1 and H2 looks very much alike, and it doesn't seem like removing the 2 outliers from training set did any significant impact on the performance. In my opinion, this could be explained by the fact that those two samples weren't significant enough out of the 365 data points in the training set, and therefore removing them wasn't as impactful to the model's performance as I had hoped it to be.

Hypothesis 3

After some experimentation with the data I wish to feed to the model in hypothesis 1 and 2, I decided to try and improve the architecture of the models based on their performance with the given data.

A - Age regression

In hypothesis 2, although the age regression performed best so far, there was a slight issue with the graph of the training, that showed almost no improvement on the validation set:



I therefore decided to try and reduce the learning rate, and change an activation function to ensure that the train loss and test loss are closer to each other and reducing together, and that the training process looks similar to the graph shown in class.

Experiment

The experiment included trying various values for the learning rate, as well as a few different activation functions, using the same dataloader used for H2. The goal of using the same dataloader is to make sure the only thing that changes is the architecture and hyperparams, while the data stays exactly the same.

Expected evidence:

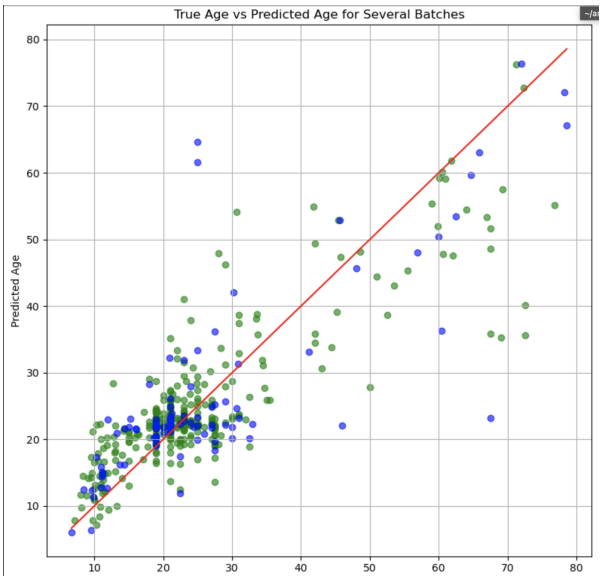
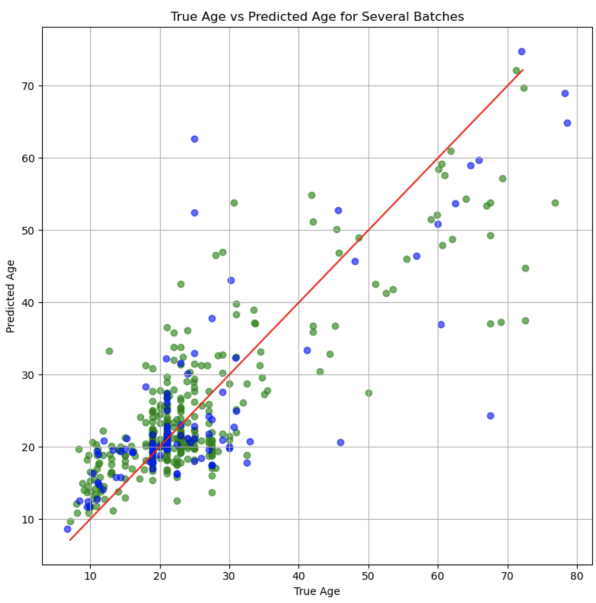
In order for me to confirm the chosen values of the hypothesis, I would expect to see evidence of lower validation error, and hopefully smoother learning graph, or training error that would be more similar to the validation error.

Results:

After trying a few alternative, the one that worked best was Leaky Relu activation function, learning rate of 3*(0.0001). This provided a smoother graph, where the results of train and test seemed to be closer to each other. However, the MAE of H2 still remained the best

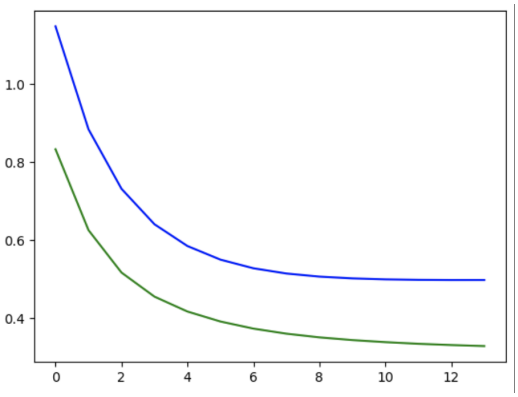
Base	**H1**	**H2**	**H3**	**Comparison**
0.537097	0.537016	0.277760	0.328128	**MSE on Train**
0.577078	0.611833	0.498469	0.497490	**MSE on Test**
6.916039	6.859985	5.839305	6.097551	**MAE on Test**

Plots

MLP H2	MLP H3
	

Discussion:

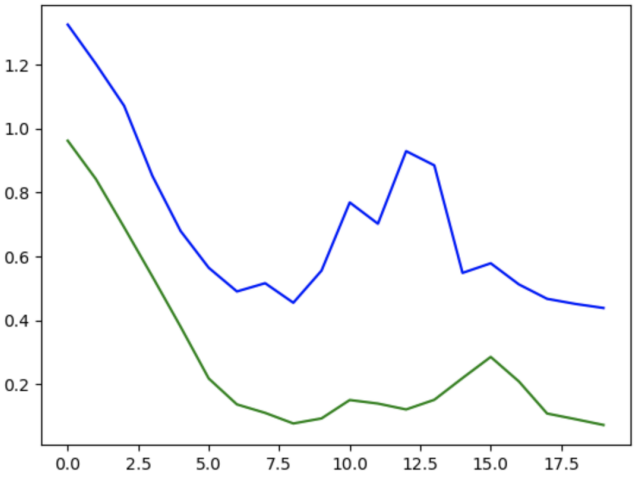
Despite trying various architectures and activation functions, I wasn't able to improve the MAE of the H2 model, which is the default architecture given in the base model part. However, the MSE loss on validation set of the model I suggested for H3 is slightly better than the previous models. There are two evidence that the H3 mlp model might be a little bit less over-fitting to the training data: Firstly, its MSE losses for training and validation are closer than those of previous models. Secondly, the training graph looks better, and indicates the model was able to learn and generalize during the training process, as opposed to the similar graph of H2 which was one of the motivations for this hypothesis:



B - Volume Prediction

Although the results of H2 training process were quite an improvement compared to the baseline model, it still uses the very basic architecture of only one convolutional layer with activation and pooling, and on linear layer at the end. Looking at the training process of this model, it looked like there is room for improvement in the ability of the model to both learn and generalize to unseen data:

H2 model - Training and Validation MSE loss for epoch - Motivation



Hypothesis:

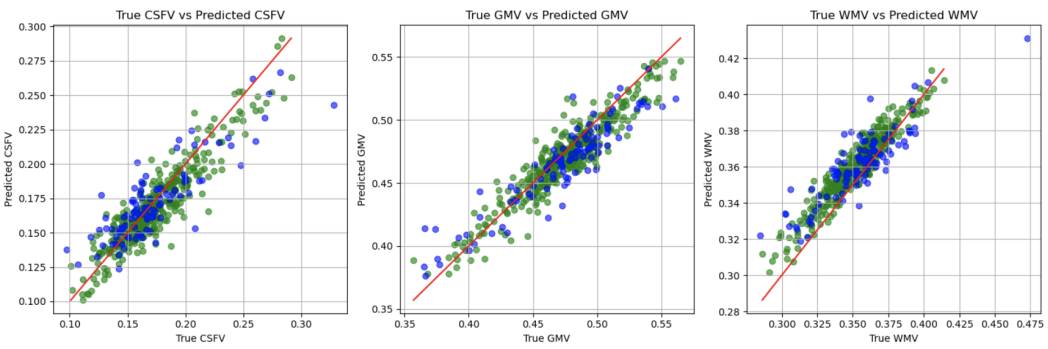
Enhancing the architecture of the model, adding convolutional layers, and an mlp head, could improve the model's ability to learn on training set and generalize on validation set.

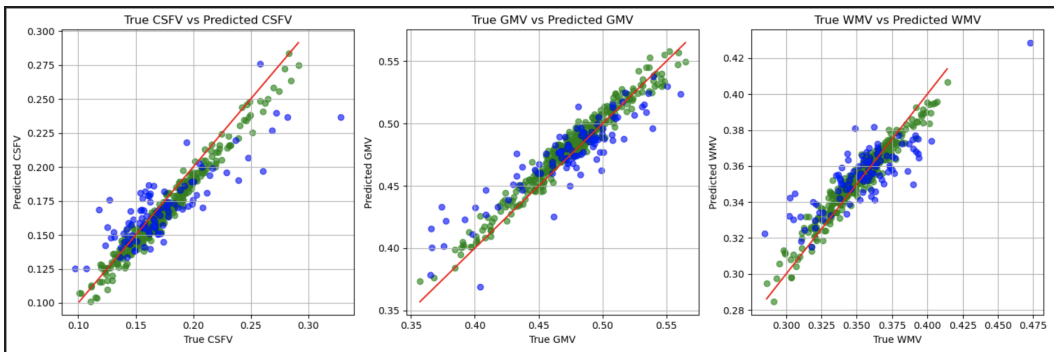
Experiment:
Similarly to the age regression, the experiment included using various deeper architectures, while using the same dataloader from H2. I decided to try adding another convolutinal layer with activation and average pooling, as well as add an mlp head to help the model learn non-linear patterns detected by the convolutional layers.

Expected evidence:
To confirm the hypothesis, I would expect to see the volume model's prediction have lower error on validation set than the error in previous hypothesis. Additionally, I think that a training process that is more similar to the graphs we saw in class could be a good indication for the successful learning of the model.

Results

Base	**H1**	**H2**	**H3**	**Comparison**
0.084129	0.105431	0.072922	0.148053	**MSE on Train**
1.041788,	0.403257	0.438615	0.374431	**MSE on Validation**
Not Relevant	0.014638	0.015278	0.013821	**MAE on Validation**

H3 model	

H2 model	
	Note: green dots represent training set, blue dots represent validation

Discussion:

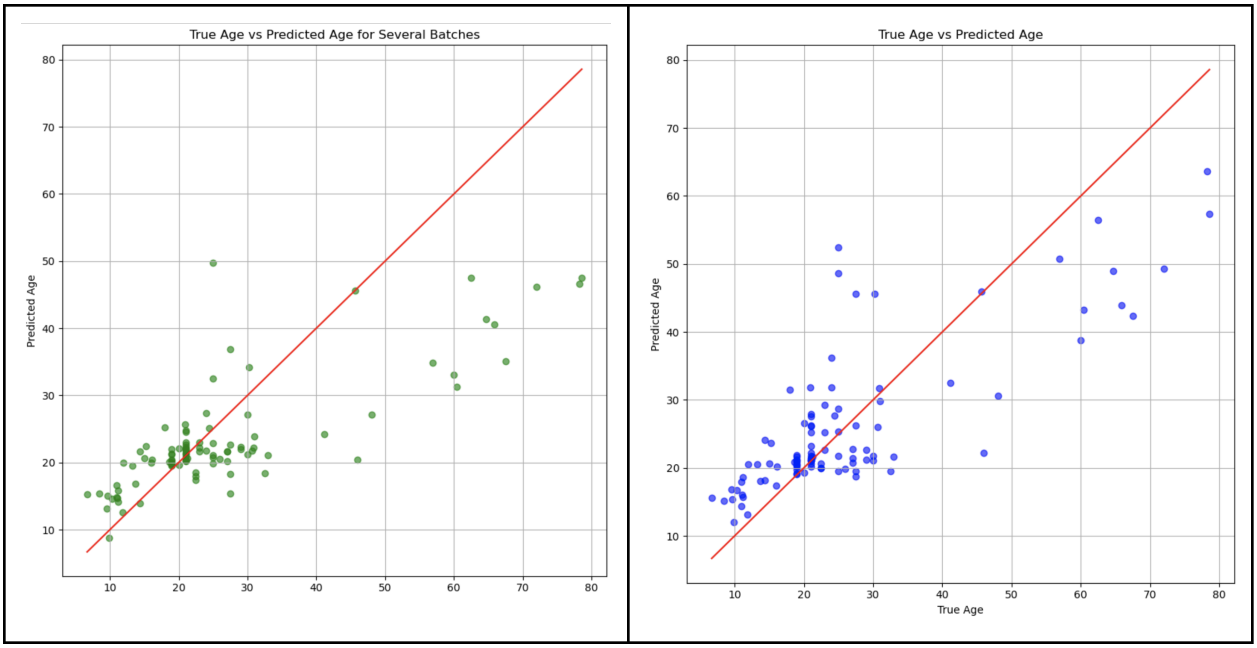
Interestingly, the volume model was able to generalize better, and it exhibits less overfitting to training set than the previous models. The first evidence for that is both in the losses, where H3 model has the highest error on training set, and the lowest error on validation set. Another evidence for the model's better ability to generalize is shown in the plots above: we can see that although the green dots (train data) are a bit further from the truth in H3 model, the blue ones (validation data) seems to be more blend in with the training data, and closer to the red line that represents the ground truth.

Combined model

Base	**H1**	**H2**	**H3**	**Comparison**
1.089519	0.685904	0.682367	0.508207	**MSE on Validation (normalized data)**
8.364075	7.012982	7.082492	6.753403	**MAE on Validation (denormalized)**

Compare plots with the most successful model so far, H1, which was quite similar to H2's plot:

H1	



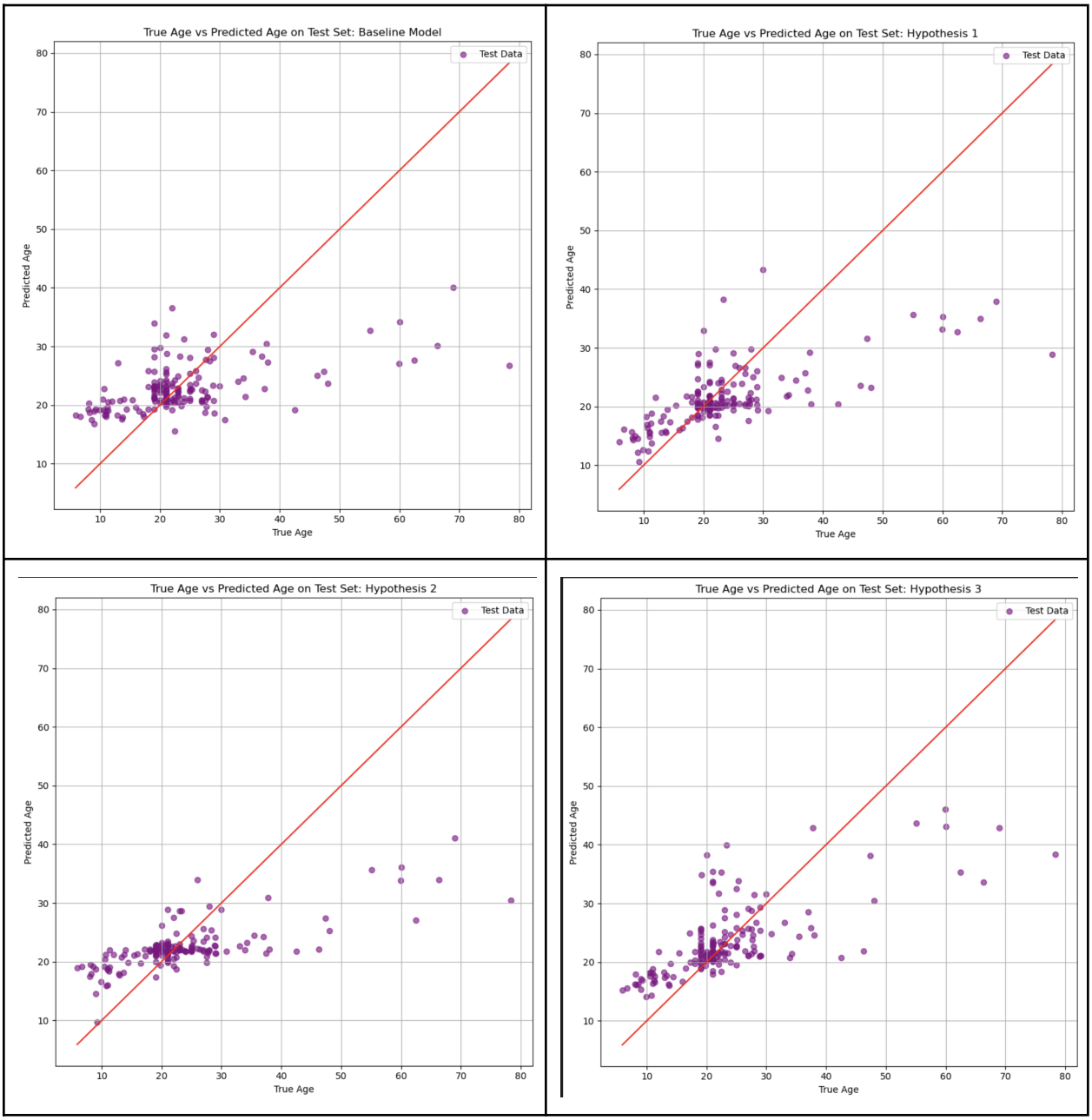
The main improvement in the results compared to previous result, is that the model predicts more accurately ages of older patients, and the ranges of its predictions is slightly larger. However, the model also predict younger patients to be older. For example, it predicts way more patients in their twenties to be over 40 years old.

Exercise 3.4: Concluding Discussion

Firstly, attached a numerical comparison between the results of all models on test data:

Metric	Base	H1	H2	H3
MAE (after denormalization)	6.325179	5.504141	5.702138	5.404752
MSE (before denormalizeaion)	0.538479	0.446839	0.454672	0.377469

Secondly, attached are the plots of the predictions of all models on test data:



To conclude, the model that performs best on both the test set and the validation set is the 3rd hypothesis: this hypothesis combines both the conversion of the volumes to proportion of each volume out of the volume's sum, cleaning outliers, and enhancing the model's architecture and learning rate. The model accepts an MRI image of a patient's brain, and provide a prediction of the patient's biological age, with a mean error of 5.4 years. However, it is important to note that the model makes large errors especially on patients over 40 years old. This may be related to the fact that the training (and validation, and test) data is skewed and contain mainly young patients.

CW2 - Coding Part 2 (30 points)

Variational Autoencoders

Build a Convolutional Variational AutoEncoder and achieve best possible reconstruction and latent space disentanglement. Then answer the questions.

In [1]:
```
import os
import numpy as np
import torch
import torch.nn as nn
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
from torchvision.utils import make_grid
from torchsummary import summary

import matplotlib.pyplot as plt
import pandas as pd
import altair as alt
```

```
/Users/user/anaconda3/lib/python3.11/site-packages/torchvision/io/image.py:
13: UserWarning: Failed to load image Python extension: 'dlopen(/Users/use
r/anaconda3/lib/python3.11/site-packages/torchvision/image.so, 0x0006): Sym
bol not found: __ZN3c106detail19maybe_wrap_dim_slowIxEET_S2_S2_b
  Referenced from: <E03EDA44-89AE-3115-9796-62BA9E0E2EDE> /Users/user/anaco
nda3/lib/python3.11/site-packages/torchvision/image.so
  Expected in:     <F2FE5CF8-5B5B-3FAD-ADF8-C77D90F49FC9> /Users/user/anaco
nda3/lib/python3.11/site-packages/torch/lib/libc10.dylib'If you don't plan
on using image functionality from `torchvision.io`, you can ignore this war
ning. Otherwise, there might be something wrong with your environment. Did
you have `libjpeg` or `libpng` installed before building `torchvision` from
source?
  warn(
```

In [2]:
```
# Use GPU if available
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")
```

```
Using cpu device
```

In [3]:
```
def show(img):
    npimg = img.cpu().numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)))
```

In [4]:
```
# Load the data

################################################################################
#                       ** START OF YOUR CODE **
################################################################################

#                 ** MODIFY CODE HERE IF NECESSARY **

batch_size = 100

data_transforms = transforms.Compose(
    [
        transforms.ToTensor(),
```

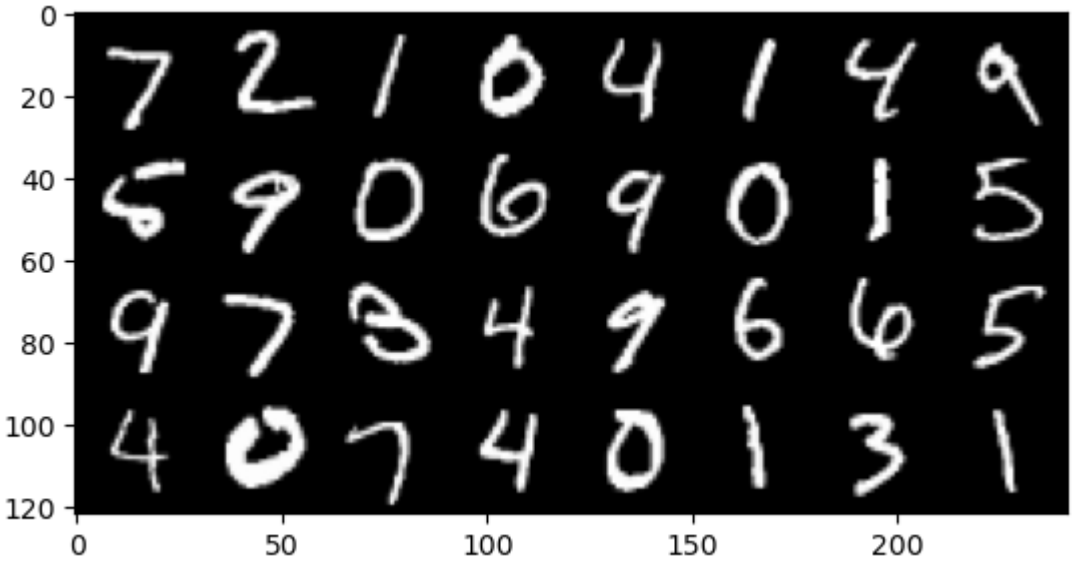
```
    ]
)

def denormalize(x):
    return x

################################################################################
#                       ** END OF YOUR CODE **
################################################################################

training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=data_transforms,
)

train_dataloader = torch.utils.data.DataLoader(
    training_data,
    batch_size=batch_size,
    shuffle=True,
)

# Download test data
test_data = datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=data_transforms,
)

test_dataloader = torch.utils.data.DataLoader(
    test_data,
    batch_size=batch_size,
)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
data/MNIST/raw/train-images-idx3-ubyte.gz
  0%|          | 0/9912422 [00:00<?, ?it/s]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
data/MNIST/raw/train-labels-idx1-ubyte.gz
  0%|          | 0/28881 [00:00<?, ?it/s]
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to d
ata/MNIST/raw/t10k-images-idx3-ubyte.gz
  0%|          | 0/1648877 [00:00<?, ?it/s]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to d
ata/MNIST/raw/t10k-labels-idx1-ubyte.gz
  0%|          | 0/4542 [00:00<?, ?it/s]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

In [5]:

```
sample_inputs, _ = next(iter(test_dataloader))
fixed_input = sample_inputs[0:32, :, :, :]
# visualize the original images of the last batch of the test set
```

```
img = make_grid(denormalize(fixed_input), nrow=8, padding=2, normalize=False
                scale_each=False, pad_value=0)
plt.figure()
show(img)
```



Variational Auto Encoders (VAEs)

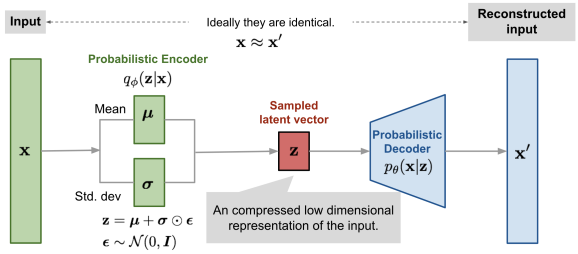


Fig.1 - VAE Diagram (with a Guassian prior), taken from 2.

Build a convolutional VAE (5 points)

The only requirement is that it contains convolutions both in the encoder and decoder. You can still use some linear layers if needed.

In [350…

```
# Convolutional VAE implementation here
class VAE(nn.Module):
    def __init__(self, latent_dim):
        super(VAE, self).__init__()
        ########################################################################
        #                        ** START OF YOUR CODE **
        ########################################################################
        self.latent_dim = latent_dim

        # Encoder layers:
        # First - 2 convolutional relu average pool layers
        self.enc_conv_relu_pool_stack = nn.Sequential(
            # layer 1
            nn.Conv2d(in_channels=1, out_channels=16, kernel_size=4, stride=
            nn.ReLU(),
            nn.AvgPool2d(2),
```

```
            # layer 2
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=4, stride
            nn.ReLU(),
            )

        # Second - MLP head with 2 fully connected layers, and Relu activati
        self.encoder_linear_1 = nn.Linear(32 * 9 * 9, 128)
        self.relu = nn.ReLU()
        self.encoder_linear_2_mean = nn.Linear(128, self.latent_dim)
        self.encoder_linear_2_std = nn.Linear(128, self.latent_dim)


        # Decoder layers:
        # First 2 fully connected layers (will be used with reluc activation
        self.decoder_linear_1 = nn.Linear(self.latent_dim, 128)
        self.decoder_linear_2 = nn.Linear(128, 16 * 10 * 10)

        # Second - 2 resize and convolutional layers.
        self.decoder_resize_conv_stack = nn.Sequential(
            # Layer 1
            transforms.Resize((18, 18), interpolation=transforms.Interpolati
            nn.Conv2d(16, 32, 4, 1, 1),
            nn.ReLU(),

            # Layer 2
            transforms.Resize((28, 28), interpolation=transforms.Interpolati
            nn.Conv2d(32, 1, 3, 1, 1),
            # Lastly, sigmoid activation for output between 0 and 1
            nn.Sigmoid(),
        )

        ########################################################################
        #                       ** END OF YOUR CODE **
        ########################################################################

    def encode(self, x):
        ########################################################################
        #                       ** START OF YOUR CODE **
        ########################################################################
        # First - convolutional layers to extract features from images
        x = self.enc_conv_relu_pool_stack(x)

        # reshape to prepare for mlp head
        x = x.view(x.size(0), -1)

        # mlp head of 2 fully connected layers and relu activation funciton
        x = self.relu(self.encoder_linear_1(x))
        mean = self.encoder_linear_2_mean(x)

        # Encoded as the log of the variance as suggested in class
        logvar = self.encoder_linear_2_std(x)

        return mean, logvar

        ########################################################################
        #                       ** END OF YOUR CODE **
        ########################################################################

    def reparametrize(self, mean, logvar):
        ########################################################################
        #                       ** START OF YOUR CODE **
        ########################################################################

        # calculate std (root quare of variance) based on log of the varianc
```

```
        std = torch.exp(0.5*logvar)

        # The prior is assumed to be distributed like a Normal Gaussian
        epsilon = torch.randn(std.size()).to(device)
        return mean + std*epsilon

        ##########################################################################
        #                        ** END OF YOUR CODE **
        ##########################################################################

    def decode(self, z):
        ##########################################################################
        #                        ** START OF YOUR CODE **
        ##########################################################################

        # First two fully connected layers with relu activation to increase
        out = self.relu(self.decoder_linear_1(z))
        out = self.relu(self.decoder_linear_2(out))

        # Reshape to 16 channels of 10*10 images, to prepare for convolutio
        out = out.view(out.size(0), 16, 10, 10)

        # Convolution and resize layers to increase resolution and reduce t
        out = self.decoder_resize_conv_stack(out)

        return out

        ##########################################################################
        #                        ** END OF YOUR CODE **
        ##########################################################################

    def forward(self, x):
        ##########################################################################
        #                        ** START OF YOUR CODE **
        ##########################################################################

        mean, logvar = self.encode(x)

        # Use reparametirization trick before reconstruction to allow backwa
        z = self.reparametrize(mean, logvar)

        # Return latent mean and logvar as well as the reconstructed image
        return mean, logvar, self.decode(z)

        ##########################################################################
        #                        ** END OF YOUR CODE **
        ##########################################################################


#############################################################################
#                        ** START OF YOUR CODE **
#############################################################################

latent_dim = 28

#############################################################################
#                        ** END OF YOUR CODE **
#############################################################################

temp_model = VAE(latent_dim).to(device)
summary(temp_model, (1, 28, 28))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 25, 25]             272
              ReLU-2          [-1, 16, 25, 25]               0
         AvgPool2d-3          [-1, 16, 12, 12]               0
            Conv2d-4            [-1, 32, 9, 9]           8,224
              ReLU-5            [-1, 32, 9, 9]               0
            Linear-6                 [-1, 128]         331,904
              ReLU-7                 [-1, 128]               0
            Linear-8                  [-1, 28]           3,612
            Linear-9                  [-1, 28]           3,612
           Linear-10                 [-1, 128]           3,712
             ReLU-11                 [-1, 128]               0
           Linear-12                [-1, 1600]         206,400
             ReLU-13                [-1, 1600]               0
           Resize-14          [-1, 16, 18, 18]               0
           Conv2d-15          [-1, 32, 17, 17]           8,224
             ReLU-16          [-1, 32, 17, 17]               0
           Resize-17          [-1, 32, 28, 28]               0
           Conv2d-18           [-1, 1, 28, 28]             289
          Sigmoid-19           [-1, 1, 28, 28]               0
================================================================
Total params: 566,249
Trainable params: 566,249
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.62
Params size (MB): 2.16
Estimated Total Size (MB): 2.79
----------------------------------------------------------------
```

Briefly Explain your architectural choices

YOUR ANSWER

The main choice to explain is the one I did for the decoder: Instead of using deconvolution layers (transposed convolution) to increase the resolution of the image, I chose to use upsampling with nearest-neighbor interpulation. This is in order to avoid the "checkboard artifact" phenomena, created by overlap in deconvolution, as detailed in this paper

Other than that, I chose to use two convolutional layers with relu activation function for the encoder, including 32 channels to allow the model learn and encode the different pattenrs that creates the different digits. I added an MLP head of 2 layers to the model to further decrease the dimention of the information into the latent random variable.

For the decoder, I started with an MLP of 2 layers with relu activation function in order to first increase the dimention of the latent vector into a dimention that could be reshaped into an image with a few channels. Then, I used layers of upsampling and regular convolution as detailed above.

In addition, I used sigmoid activation function on the last layer of the decoder, to ensure that the output will be between 0 and 1, similarly to the input domain. This will be useful for the selection of loss function, as detailed in next question.

Defining a Loss (6 points)

The Beta VAE loss, with encoder q and decoder p:

$$L = \mathbb{E}_{q_\phi(z|X)}[\log p_\theta(X \mid z)] - \beta D_{KL}[q_\phi(z \mid X) \| p_\theta(z)]$$

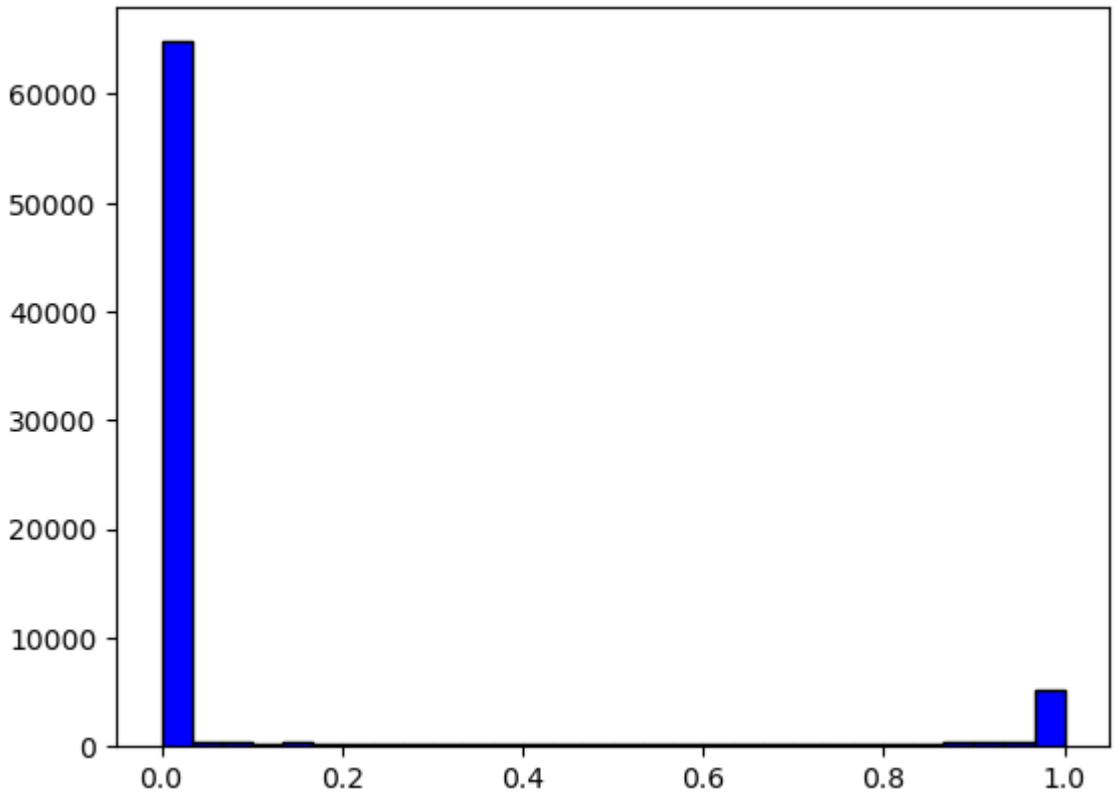
The loss you implement depends on your choice of latent prior and model outputs.

There exist different solutions that are equally correct. Depending on your assumptions you might want to do a data preprocessing step.

In [224…
```
# Data exploration for deciding on loss function -
# Firstly, check the values of the pixels to understand the nature of the da
# If the values are between 0 and 1, it might be worth to use the cross enti
sample_input, _ = next(iter(test_dataloader))
all_pixels_vals = torch.flatten(sample_input)
print(f"max_val is: {all_pixels_vals.max()}")
print(f"min val is: {all_pixels_vals.min()}")
print(f"mean is: {all_pixels_vals.mean()}")
print(f"std is: {all_pixels_vals.std()}")
plt.hist(all_pixels_vals, bins=30, color='blue', edgecolor='black')
```

```
max_val is: 1.0
min val is: 0.0
mean is: 0.11988329887390137
std is: 0.2958832383155823
```

Out[224]:
```
 (array([64814.,   431.,   474.,   311.,   339.,   267.,   304.,   268.,
           253.,   232.,   286.,   172.,   268.,   259.,   225.,   321.,
           246.,   246.,   257.,   238.,   258.,   256.,   274.,   272.,
           314.,   288.,   369.,   410.,   465.,  5283.]),
  array([0.        , 0.03333334, 0.06666667, 0.1       , 0.13333334,
         0.16666667, 0.2       , 0.23333333, 0.26666668, 0.30000001,
         0.33333334, 0.36666667, 0.40000001, 0.43333334, 0.46666667,
         0.5       , 0.53333336, 0.56666666, 0.60000002, 0.63333333,
         0.66666669, 0.69999999, 0.73333335, 0.76666665, 0.80000001,
         0.83333331, 0.86666667, 0.89999998, 0.93333334, 0.96666664,
         1.        ]),
  <BarContainer object of 30 artists>)
```



In [381…
```
## Explore binary cross entropy values for better understanding and for driv
prediction = torch.tensor([0.0])
target = torch.tensor([1.0])

bce_loss = nn.functional.binary_cross_entropy(prediction, target)
print(bce_loss.item())

prediction = torch.tensor([0.2])
target = torch.tensor([0.3])

bce_loss = nn.functional.binary_cross_entropy(prediction, target)
print(bce_loss.item())
```

```
100.0
0.6390318870544434
```

In [235…
```
def loss_function_VAE(recon_x, x, mu, logvar, beta):
        ###################################################################
        #                       ** START OF YOUR CODE **
        ###################################################################
        cross_antropy_loss = nn.functional.binary_cross_entropy(input=recon_
        KL_divergence = 0.5*torch.sum(1+logvar-mu**2-torch.exp(logvar))

        return cross_antropy_loss - beta * KL_divergence

        ###################################################################
        #                       ** END OF YOUR CODE **
        ###################################################################
```

Briefly answer the following:

a. Explain what are the possible choices of reconstruction loss, and which one you choose. Explain how it relates to

1. Your choice of VAE prior
2. The output data domain

3. The latent space disentanglement

Feel free to try and train with different reconstruction losses to see which one works best

YOUR ANSWER

a. The possible choices for reconstruction loss are Mean Square Error, and Binray Cross Entropy error. I chose to use the Binary Cross Entropy error since I looked at the distribution of the data, and saw that

- all pixels are between 0 and 1
- the vast majority of the pixels are either 0 or 1, and only a small fraction of them are values in between.

I wanted to choose error function such that the loss for a pixel with value 0 that was reconstructed as 1 will be maximal, while the loss on a grey pixel that was reconstructed as a different shade of grey will be lower. The cross entropy fits right into these requirements, as it is not capped by 1 like MSE in this case.

1. My choice of VAE prior is related to the calculation of the kl divergence component in the regularization term. The formula I used to calculate the kl divergence assumes the latent variable is normally distributed, with means in the vector mu and log of the variance in the vector logvar.
2. The choise is related to the output data domain, as the binary cross entropy assumes that all the outputs of the decoder (as well as inputs of the encoder) will be composed of values between 0 and 1. This is a valid assumption for me to make, as my model applies sigmoid activation function on the output just before returning it, which ensures all the pixels will be in the relevant range. It is also important to note that the input domain includes pixels that are normalized between 0 and 1, as shown in the distribution graph above. This is relevant since the input is used as the target for calculating the loss, and thus must be capped between 0 and 1 in order to allow using the cross entropy loss.
3. The latent space disentanglement is enforced by the regularization term, which compels the model to only utilize dimensions in the latent variable if they contribute valuable information. It penalizes the model for using unnecessary dimensions in the latent vector. Since the values of binary cross-entropy are typically larger than the values of MSE, the regularization term becomes less significant in the loss. This may require trying a larger beta than the beta that would have suited the MSE.

Train and plot

Train the VAE and plot:

1. The total loss curves for train and test (on the same plot)
2. The reconstruction losses for train and test (on the same plot)
3. The KL losses for train and test (on the same plot)

(x-axis: epochs, y-axis: loss)

You may want to have different plots with differente values of β.

Hint: You can modify the training scripts provided in previous tutorials to record the required information, and use matplotlib to plot them Hint: If you plan on doing hyperparameter tuning, it might be a good idea to split the training set and create a validation set

1. Defining methods for training and testing

The methods not only train and test, but also calculate and return the total loss, kl loss and reconstruction loss

In [262…

```
# Training code

#################################################################################
#                        ** START OF YOUR CODE **
#################################################################################

# Define seperated function to calculate the different components of the los
def reconstruction_loss(recon_x, x):
    return torch.pow(recon_x - x,2).sum()

def kl_divergence_loss(mu, logvar):
    kl_divergence = 0.5*torch.sum(1+logvar-mu**2-torch.exp(logvar))
    return kl_divergence


def train_one_epoch(dataloader, model, loss_fn, recon_loss_fn, kl_fn, optim
    """
    Train the model on one epoch, and return the three losses.
    mean_loss - The full loss comprised of reconstruction loss - beta * kl_
    mean_recon_loss - only the reconstruciton loss part
    mean_kl_divergance - the kl_divergence part of the loss, not multiplied
    """
    size = len(dataloader.dataset)
    model.train()
    total_loss = 0
    total_recon_loss = 0
    total_kl_divergance = 0
    for batch, (x, _) in enumerate(dataloader):

        x = x.to(device)

        # Compute prediction error
        mu, logvar, recon_x = model(x)
        loss = loss_fn(recon_x, x, mu, logvar, beta)
        total_loss += loss.item()

        recon_loss = recon_loss_fn(recon_x, x)
        total_recon_loss += recon_loss.item()

        kl_divergance_loss = kl_fn(mu, logvar)
        total_kl_divergance += kl_divergance_loss.item()


        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
    mean_loss = total_loss / size
    mean_recon_loss = total_recon_loss/size
    mean_kl_divergance = total_kl_divergance / size
    if should_print_loss:
        print(f"training loss: {mean_loss:>5f}")
    return mean_loss, mean_recon_loss, mean_kl_divergance


def test(dataloader, model, loss_fn, recon_loss_fn, kl_fn, beta, should_prin
    """
    Tests the model on the given dataloader, and return the three losses.
    total_loss - The full loss comprised of reconstruction loss - beta * kl_
    total_recon_loss - only the reconstruciton loss part
    total_kl_loss - the kl_divergence part of the loss, *not multiplied by b
    """
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    total_loss = 0
    total_recon_loss = 0
    total_kl_loss = 0
    with torch.no_grad():
        for x, _ in dataloader:
            x = x.to(device)

            mu, logvar, recon_x = model(x)

            loss = loss_fn(recon_x, x, mu, logvar, beta)
            total_loss += loss.item()

            recon_loss = recon_loss_fn(recon_x, x)
            total_recon_loss += recon_loss.item()

            kl_loss = kl_fn(mu, logvar)
            total_kl_loss += kl_loss.item()


    total_loss /= size
    total_recon_loss /= size
    total_kl_loss /= size
    if should_print_loss:
        print(f"test loss: {total_loss:>8f} \n")
    return total_loss, total_recon_loss, total_kl_loss
```

2. Hyper parameter tuning

a. Split training set to train and validation sets

In [248…

```
# Hyper parameters tuning - prepare train and validation dataloaders

from sklearn import model_selection
rnd_seed = 42


train_split_set, validation_split_set = model_selection.train_test_split(tra

train_split_dataloader = torch.utils.data.DataLoader(
    train_split_set,
    batch_size=batch_size,
    shuffle=True,
)

validation_dataloader = torch.utils.data.DataLoader(
```

```
    validation_split_set,
    batch_size=batch_size,
    shuffle=True,
)
```

2. Hyper parameter tuning

b. Performa Grid Search to find the best hyperparams for the problem:

- Define sets of Betas an Latent dimensions to explore, then train and validate models for all possible combinations. Save the models to a pickle file for future reference and better exploration in this coursework
- Save all results for the training processes in dictionaries, for plotting them later on

In [267…

```
# Hyper parameters tuning - Train models with various latent dimentions and

import pickle
import itertools


EPOCHS = 8
LR = 0.0001

MODELS_DIR = "models"

def get_model_name(beta, latent_dim):
    return f"model_beta_{beta}_latent_dim_{latent_dim}"

def train_model_with_params(beta, latent_dim):
    model_name = get_model_name(beta, latent_dim)
    model = VAE(latent_dim).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)

    train_losses = []
    train_recon_losses = []
    train_kl_divergances = []

    val_losses = []
    val_recon_losses = []
    val_kl_divergances = []
    should_print_loss=False
    for epoch in range(EPOCHS):
        if epoch % 4 == 0:
            should_print_loss=True
            print(f"Epoch {epoch+1}\n-------------------------------")
        mean_loss_train, mean_recon_loss_train, mean_kl_divergance_train = t
        train_losses.append(mean_loss_train)
        train_recon_losses.append(mean_recon_loss_train)
        train_kl_divergances.append(mean_kl_divergance_train)

        mean_loss_test, mean_recon_loss_test, mean_kl_divergance_test = test
        val_losses.append(mean_loss_test)
        val_recon_losses.append(mean_recon_loss_test)
        val_kl_divergances.append(mean_kl_divergance_test)
        should_print_loss=False


    results_dict = {"train_total_loss":train_losses,
                    "val_total_loss": val_losses,
                    "train_recon_loss": train_recon_losses,
                    "val_recon_loss": val_recon_losses,
                    "train_kl_divergances":train_kl_divergances,
```

```
                    "val_kl_divergances": val_kl_divergances}

    last_epoch_results_dict = {"train_total_loss":mean_loss_train,
                "val_total_loss": mean_loss_test,
                "train_recon_loss": mean_recon_loss_train,
                "val_recon_loss": mean_recon_loss_test,
                "train_kl_divergances":mean_kl_divergance_train,
                "val_kl_divergances": mean_kl_divergance_test}

    # Save a trained version of the model
    # The plot might not be enough for choosing betweenthe different models,
    with open(os.path.join(MODELS_DIR, f"{model_name}.pkl"), "wb") as f:
        pickle.dump(model, f)

    return results_dict, last_epoch_results_dict



betas = [0.1, 0.5, 1, 5]
latent_dims= [10, 20, 30]

def hyperparams_selection(betas, latent_dims):
    full_training_results = {}
    final_results = {}
    hyperparams_combinations = list(itertools.product(betas, latent_dims))
    for beta, latent_dim in hyperparams_combinations:
        model_name = get_model_name(beta, latent_dim)
        print(f"Training and evalutaing {model_name}")
        models_results, model_last_epoch_results = train_model_with_params(b
        full_training_results[model_name] = models_results
        final_results[model_name] = model_last_epoch_results
        print(f"{model_name} final results:")
        print(model_last_epoch_results)

    return full_training_results, final_results

full_training_results_dict, final_results_dict = hyperparams_selection(betas
```

```
Training and evalutaing model_beta_0.1_latent_dim_10
Epoch 1
-------------------------------
training loss: 240.713553
test loss: 199.993262

Epoch 5
-------------------------------
training loss: 100.515172
test loss: 98.907665

model_beta_0.1_latent_dim_10 final results:
{'train_total_loss': 93.8019775390625, 'val_total_loss': 93.53630034722222,
'train_recon_loss': 13.869666372261795, 'val_recon_loss': 13.77946011013454
9, 'train_kl_divergances': -33.11565887331495, 'val_kl_divergances': -33.17
170518663195}
Training and evalutaing model_beta_0.1_latent_dim_20
Epoch 1
-------------------------------
training loss: 241.141213
test loss: 193.294772

Epoch 5
-------------------------------
training loss: 95.979743
test loss: 93.571442

model_beta_0.1_latent_dim_20 final results:
{'train_total_loss': 85.87362544998469, 'val_total_loss': 85.0845184461805
5, 'train_recon_loss': 10.473190967036228, 'val_recon_loss': 10.20523740641
276, 'train_kl_divergances': -55.62160302734375, 'val_kl_divergances': -55.
63289849175347}
Training and evalutaing model_beta_0.1_latent_dim_30
Epoch 1
-------------------------------
training loss: 241.340885
test loss: 189.100522

Epoch 5
-------------------------------
training loss: 93.298116
test loss: 91.131132

model_beta_0.1_latent_dim_30 final results:
{'train_total_loss': 83.66903984757965, 'val_total_loss': 82.8814922960069
4, 'train_recon_loss': 9.236105059455422, 'val_recon_loss': 8.9687823825412
33, 'train_kl_divergances': -70.93132627719056, 'val_kl_divergances': -71.0
2653537326388}
Training and evalutaing model_beta_0.5_latent_dim_10
Epoch 1
-------------------------------
training loss: 241.080171
test loss: 196.354240

Epoch 5
-------------------------------
training loss: 112.132025
test loss: 110.490487

model_beta_0.5_latent_dim_10 final results:
{'train_total_loss': 105.3790495557598, 'val_total_loss': 105.1752111545138
9, 'train_recon_loss': 15.12831658337163, 'val_recon_loss': 15.061686971028
646, 'train_kl_divergances': -22.188529986213236, 'val_kl_divergances': -2
2.254228081597223}
```

```
Training and evalutaing model_beta_0.5_latent_dim_20
Epoch 1
-------------------------------
training loss: 237.384657
test loss: 187.017195

Epoch 5
-------------------------------
training loss: 110.067218
test loss: 108.134851

model_beta_0.5_latent_dim_20 final results:
{'train_total_loss': 101.79933494178921, 'val_total_loss': 101.230285373263
88, 'train_recon_loss': 12.208261108398437, 'val_recon_loss': 11.9697338189
0191, 'train_kl_divergances': -32.57621638039981, 'val_kl_divergances': -3
2.80557264539931}
Training and evalutaing model_beta_0.5_latent_dim_30
Epoch 1
-------------------------------
training loss: 235.502958
test loss: 187.965129

Epoch 5
-------------------------------
training loss: 111.911520
test loss: 110.160348

model_beta_0.5_latent_dim_30 final results:
{'train_total_loss': 103.18519768688725, 'val_total_loss': 102.590861979166
66, 'train_recon_loss': 12.03135278080959, 'val_recon_loss': 11.78927304416
2327, 'train_kl_divergances': -36.30835419538909, 'val_kl_divergances': -3
6.40706925455729}
Training and evalutaing model_beta_1_latent_dim_10
Epoch 1
-------------------------------
training loss: 243.091034
test loss: 199.600643

Epoch 5
-------------------------------
training loss: 122.896464
test loss: 120.916176

model_beta_1_latent_dim_10 final results:
{'train_total_loss': 115.02845352711397, 'val_total_loss': 114.585799696180
56, 'train_recon_loss': 16.108228302600338, 'val_recon_loss': 15.9650826687
28298, 'train_kl_divergances': -17.807378563974417, 'val_kl_divergances': -
17.73446784125434}
Training and evalutaing model_beta_1_latent_dim_20
Epoch 1
-------------------------------
training loss: 241.710862
test loss: 195.892808

Epoch 5
-------------------------------
training loss: 126.406143
test loss: 124.125415

model_beta_1_latent_dim_20 final results:
{'train_total_loss': 116.83447457107843, 'val_total_loss': 115.962467013888
88, 'train_recon_loss': 15.14948111261106, 'val_recon_loss': 14.79710877821
1806, 'train_kl_divergances': -22.497082983877146, 'val_kl_divergances': -2
2.645473578559027}
```

```
Training and evalutaing model_beta_1_latent_dim_30
Epoch 1
-------------------------------
training loss: 241.662762
test loss: 195.165002

Epoch 5
-------------------------------
training loss: 132.661626
test loss: 130.281520

model_beta_1_latent_dim_30 final results:
{'train_total_loss': 122.58354645373774, 'val_total_loss': 121.587963324652
78, 'train_recon_loss': 16.947293732287836, 'val_recon_loss': 16.5282240532
76908, 'train_kl_divergances': -22.749068737553614, 'val_kl_divergances': -
22.888204752604167}
Training and evalutaing model_beta_5_latent_dim_10
Epoch 1
-------------------------------
training loss: 247.629771
test loss: 204.059004

Epoch 5
-------------------------------
training loss: 172.744649
test loss: 170.496841

model_beta_5_latent_dim_10 final results:
{'train_total_loss': 164.45521432674633, 'val_total_loss': 163.8050390625,
'train_recon_loss': 28.869896843405332, 'val_recon_loss': 28.38960321723090
3, 'train_kl_divergances': -5.97498731545841, 'val_kl_divergances': -6.1027
86736382378}
Training and evalutaing model_beta_5_latent_dim_20
Epoch 1
-------------------------------
training loss: 252.945264
test loss: 202.991412

Epoch 5
-------------------------------
training loss: 177.221420
test loss: 175.025430

model_beta_5_latent_dim_20 final results:
{'train_total_loss': 168.31833957567403, 'val_total_loss': 167.449080403645
84, 'train_recon_loss': 30.15983846028646, 'val_recon_loss': 29.73747444661
4584, 'train_kl_divergances': -5.9458084644990805, 'val_kl_divergances': -
5.991531182183159}
Training and evalutaing model_beta_5_latent_dim_30
Epoch 1
-------------------------------
training loss: 248.523441
test loss: 203.938942

Epoch 5
-------------------------------
training loss: 180.200432
test loss: 178.245504

model_beta_5_latent_dim_30 final results:
{'train_total_loss': 171.27127274816178, 'val_total_loss': 170.363887803819
44, 'train_recon_loss': 31.982464101753983, 'val_recon_loss': 31.5289026150
17362, 'train_kl_divergances': -5.549567579082415, 'val_kl_divergances': -
5.6082300821940105}
```

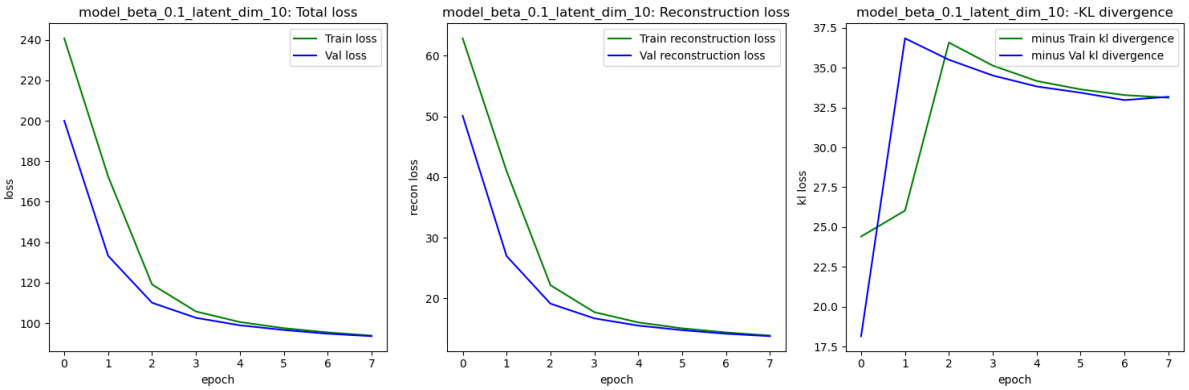
2. Hyper parameter tuning

c. Plot graphs of train and validation losses during the training process for the models trained above.

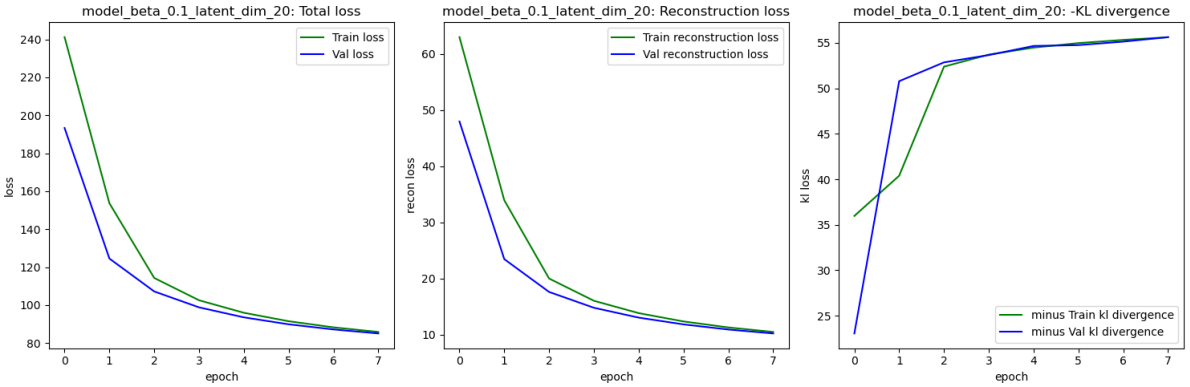
In [331…

```
def plot_model_losses(model_name, losses_dict, epochs):
    print(f"plotting losses for {model_name}")
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
    # Plot total loss
    axes[0].plot(range(epochs), losses_dict["train_total_loss"], color="gree
    axes[0].plot(range(epochs), losses_dict["val_total_loss"], color="blue",
    axes[0].set_title(f'{model_name}: Total loss')
    axes[0].set_xlabel('epoch')
    axes[0].set_ylabel('loss')
    axes[0].legend()


    # reconstruction loss
    axes[1].plot(range(epochs), losses_dict["train_recon_loss"], color="gree
    axes[1].plot(range(epochs), losses_dict["val_recon_loss"], color="blue",
    axes[1].set_title(f'{model_name}: Reconstruction loss')
    axes[1].set_xlabel('epoch')
    axes[1].set_ylabel('recon loss')
    axes[1].legend()

    # kl loss
    axes[2].plot(range(epochs), [-1 * kl for kl in losses_dict["train_kl_div
    axes[2].plot(range(epochs), [-1 * kl for kl in losses_dict["val_kl_diver
    axes[2].set_title(f'{model_name}: -KL divergence')
    axes[2].set_xlabel('epoch')
    axes[2].set_ylabel('kl loss')
    axes[2].legend()

    # Display the plots
    plt.tight_layout()
    plt.show()


## plot the training and validation loss of each of the models
for beta in betas:
    for latent_dim in latent_dims:
        model_name = get_model_name(beta, latent_dim)
        plot_model_losses(model_name, full_training_results_dict[model_name]
```

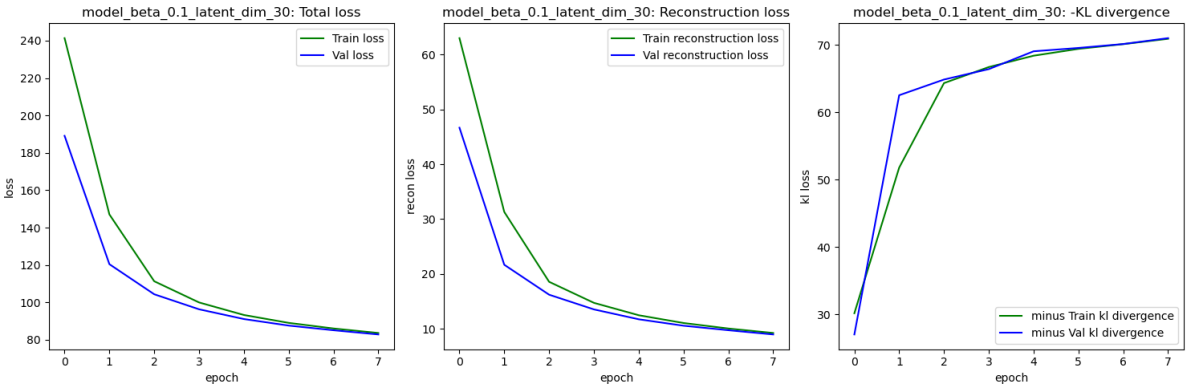
```
plotting losses for model_beta_0.1_latent_dim_10
```



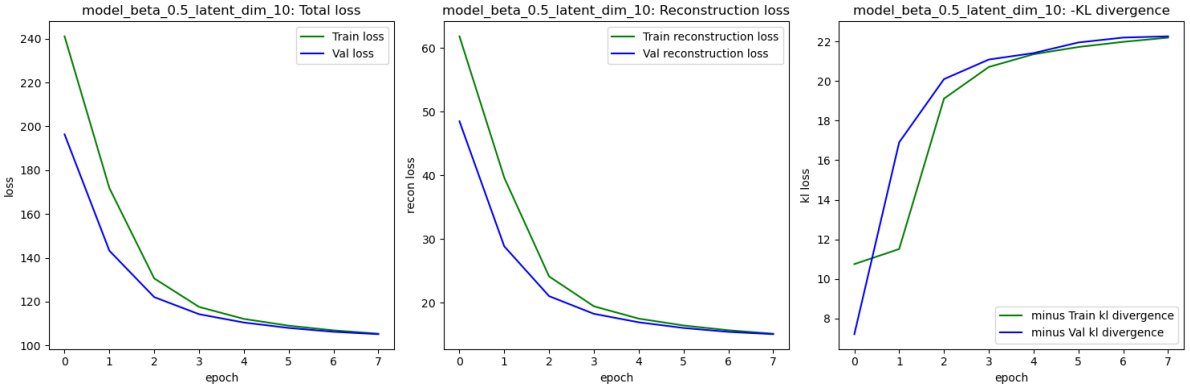

```
plotting losses for model_beta_0.1_latent_dim_20
```



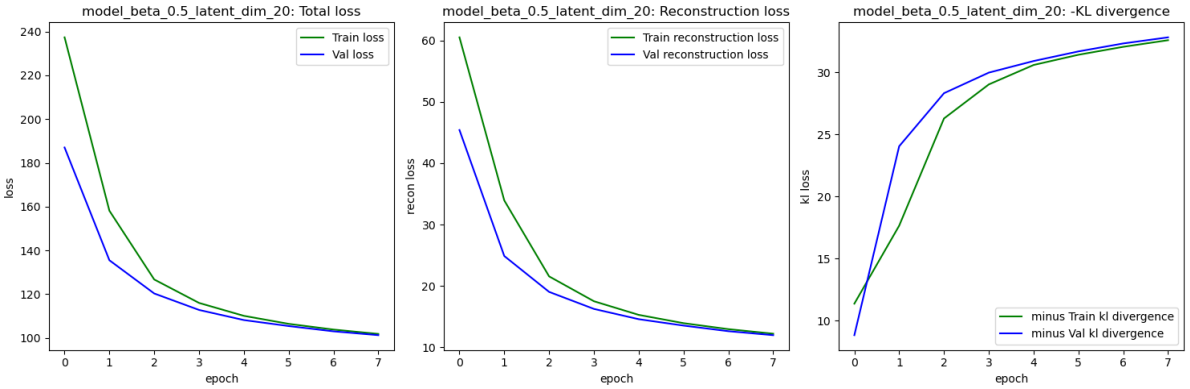
plotting losses for model_beta_0.1_latent_dim_30



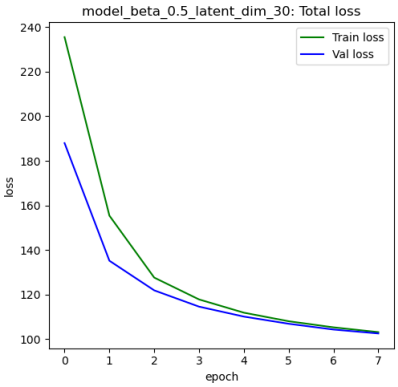
plotting losses for model_beta_0.5_latent_dim_10

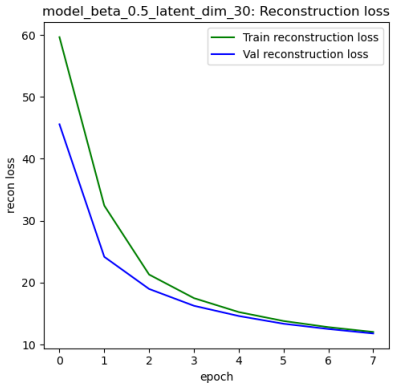


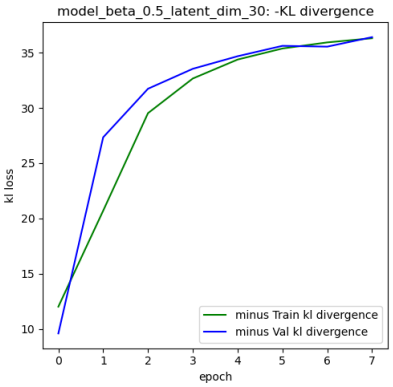
plotting losses for model_beta_0.5_latent_dim_20



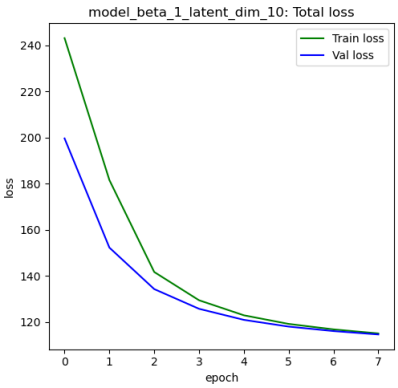
plotting losses for model_beta_0.5_latent_dim_30

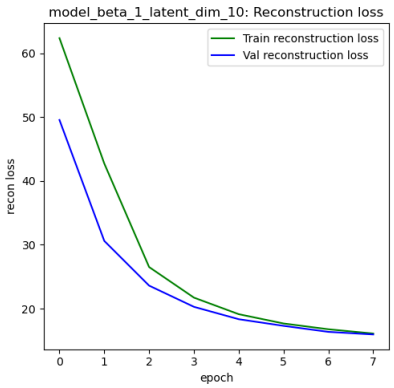


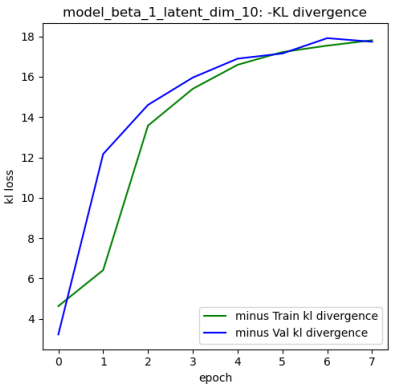




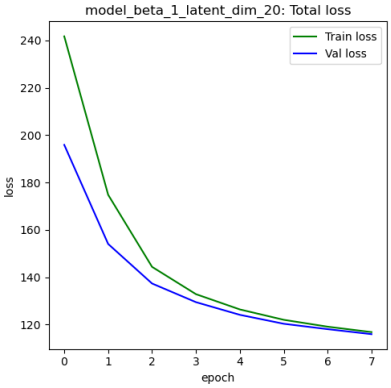
plotting losses for model_beta_1_latent_dim_10

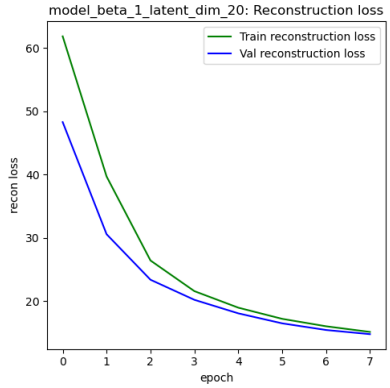


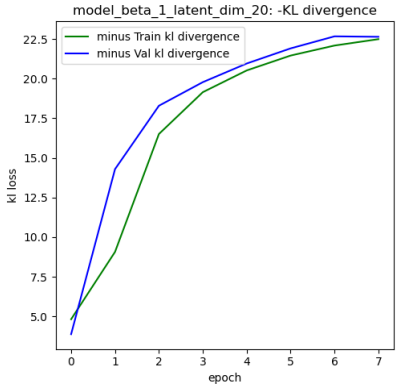




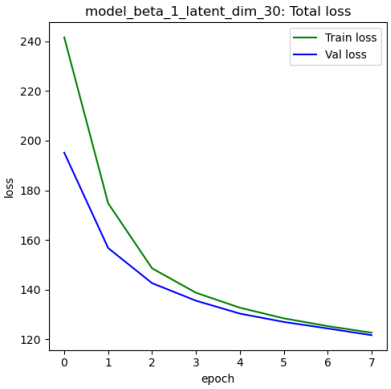
plotting losses for model_beta_1_latent_dim_20

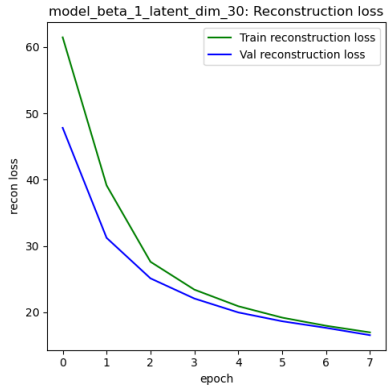


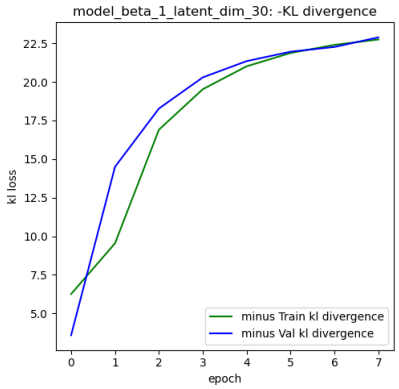




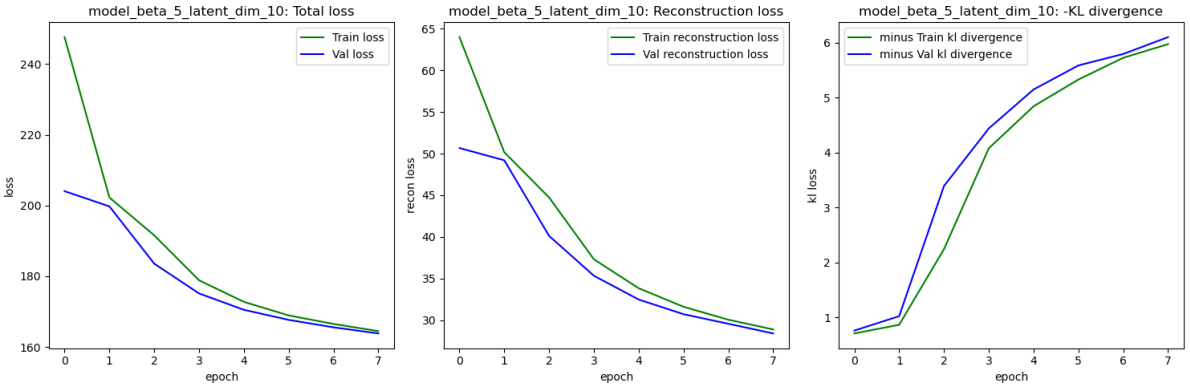
plotting losses for model_beta_1_latent_dim_30



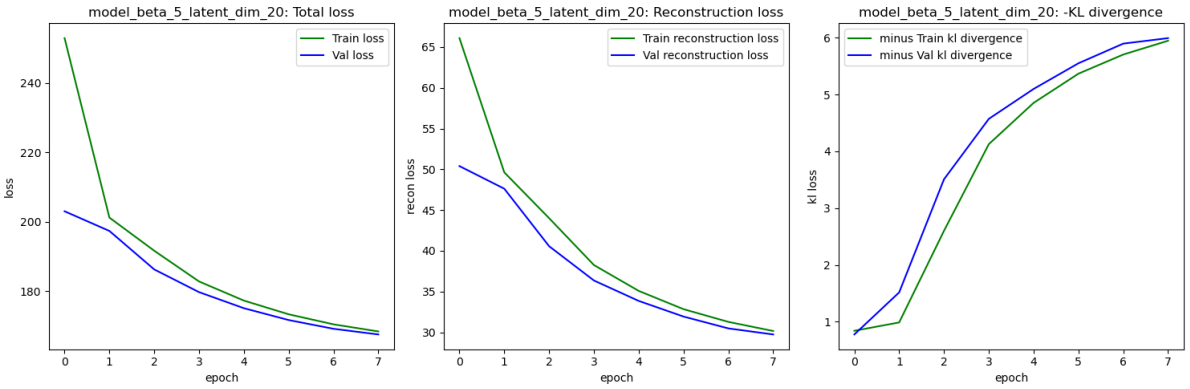




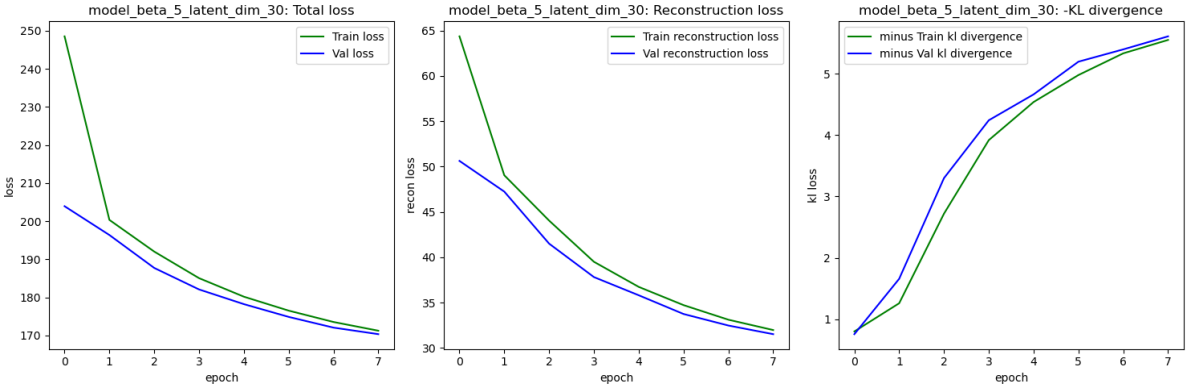
plotting losses for model_beta_5_latent_dim_10




```
plotting losses for model_beta_5_latent_dim_20
```




```
plotting losses for model_beta_5_latent_dim_30
```



Hyper Parameters Tuning

d. **Conclusions from visualization of the training process of the different models:**

- The chosen model's beta shouldn't be as low as 0.1, as this makes the kl divergence explode, especially for high latent dimensions
- The chosen model's beta shouldn't be as high as 5, as this has a bad impact on the reconstruction error, which becomes as high as 30.
- For small betas, high latent dimension increases the kl divergence (for the same beta value).
- The increase of latent value from 20 to 30 does not reduce the reconstruction error. Therefore, it is redundent to use such a high dimention as 30

Next step:

train 2 more models with beta = 0.75 for latent dimensions 10 and 20, then decide between those, and those for beta = 1, 0.5 for the same latent dimensions

In [322…

```
extra_betas = [0.75]
focused_latent_dims = [10, 20]
```

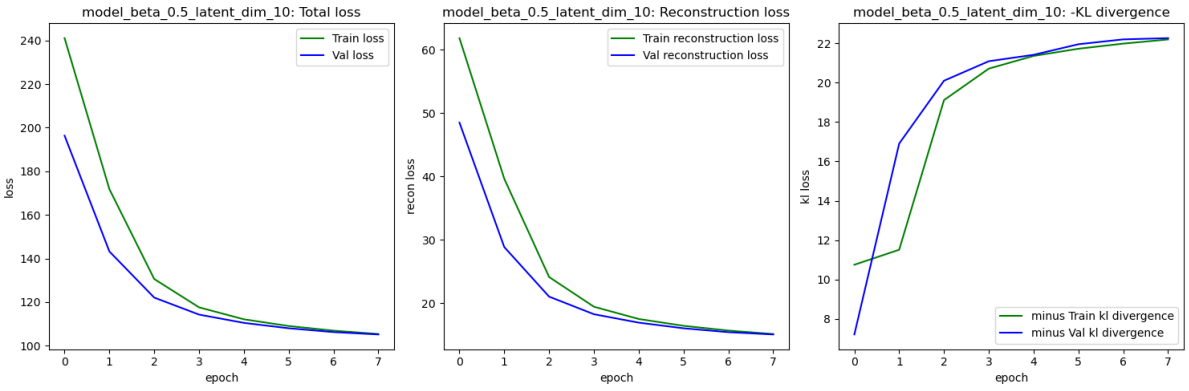
```
additional_full_training_results_dict, additional_final_results_dict = hyper
```

```
Training and evalutaing model_beta_0.75_latent_dim_10
Epoch 1
-------------------------------
training loss: 251.660987
test loss: 196.313865

Epoch 5
-------------------------------
training loss: 118.545442
test loss: 117.036640

model_beta_0.75_latent_dim_10 final results:
{'train_total_loss': 111.8925180568321, 'val_total_loss': 111.4292300347222
2, 'train_recon_loss': 16.17551154162837, 'val_recon_loss': 16.013725911458
334, 'train_kl_divergances': -19.340584218941483, 'val_kl_divergances': -1
9.36729679361979}
Training and evalutaing model_beta_0.75_latent_dim_20
Epoch 1
-------------------------------
training loss: 246.193212
test loss: 195.291753

Epoch 5
-------------------------------
training loss: 120.788198
test loss: 118.516364

model_beta_0.75_latent_dim_20 final results:
{'train_total_loss': 111.26839142922795, 'val_total_loss': 110.516176215277
78, 'train_recon_loss': 14.342555817248774, 'val_recon_loss': 14.1162714029
94792, 'train_kl_divergances': -25.664234111711092, 'val_kl_divergances': -
25.505162489149306}
```

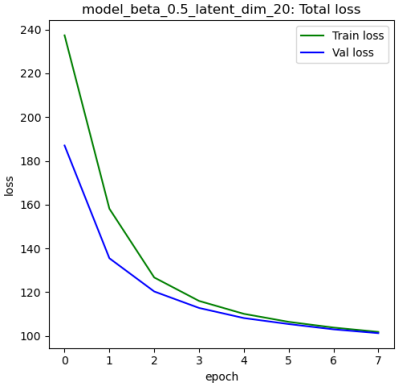
In [332…

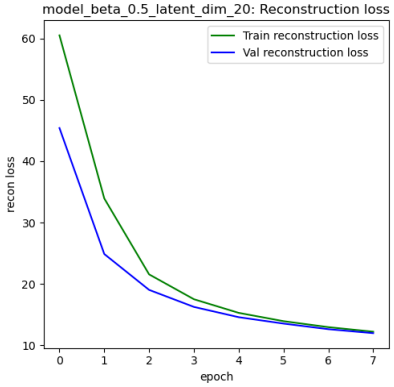
```
betas_for_decision = [0.5, 0.75, 1]
dims_for_decision = [10, 20]

models_results_for_decision = additional_full_training_results_dict.copy()

for beta in [0.5, 1]:
    for dim in dims_for_decision:
        model_name = get_model_name(beta, dim)
        models_results_for_decision[model_name] = full_training_results_dict

hyperparams_combinations = list(itertools.product(betas_for_decision, dims_
for beta, dim in hyperparams_combinations:
    model_name = get_model_name(beta, dim)
    plot_model_losses(model_name, models_results_for_decision[model_name], E
```

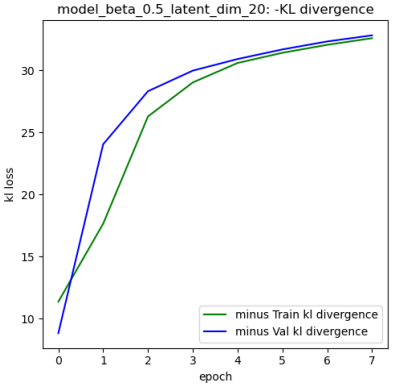
```
plotting losses for model_beta_0.5_latent_dim_10
```



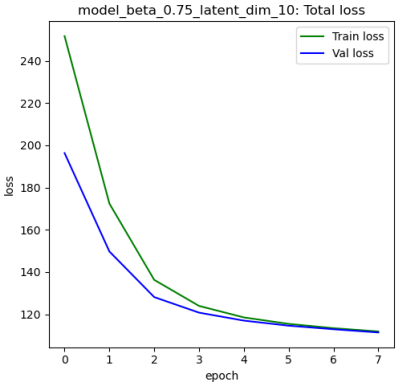
plotting losses for model_beta_0.5_latent_dim_20

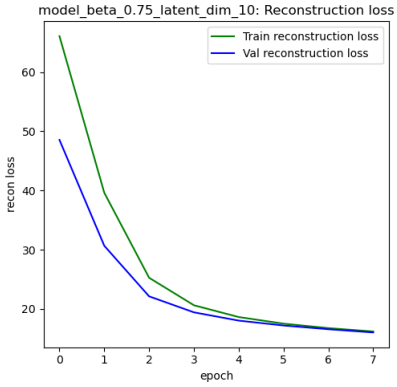


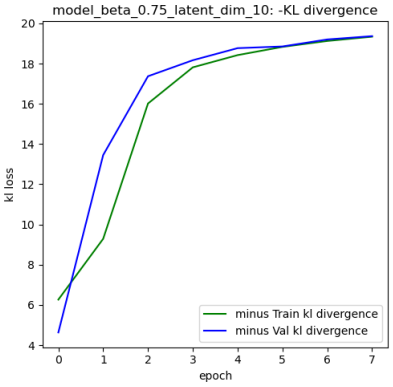




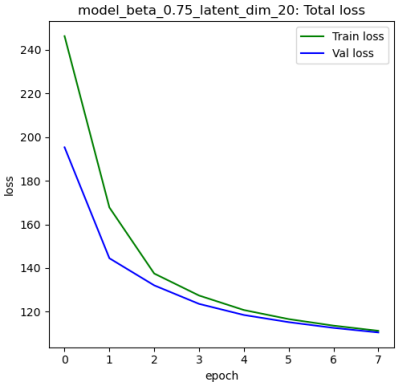
plotting losses for model_beta_0.75_latent_dim_10

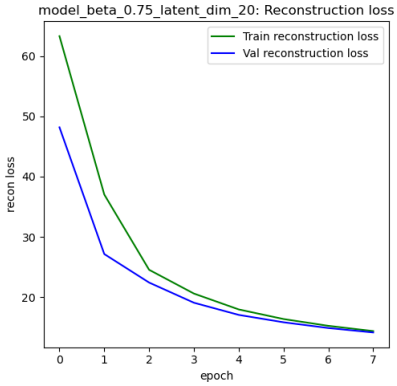


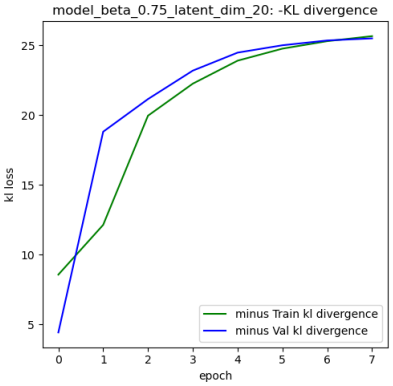




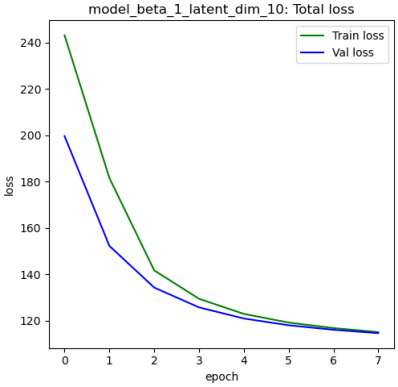
plotting losses for model_beta_0.75_latent_dim_20

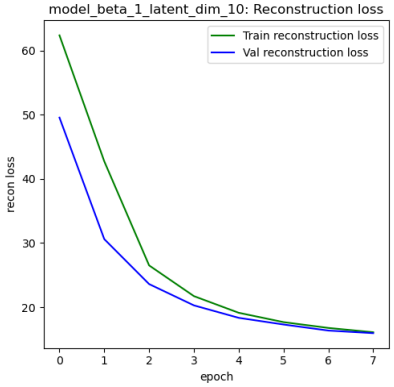


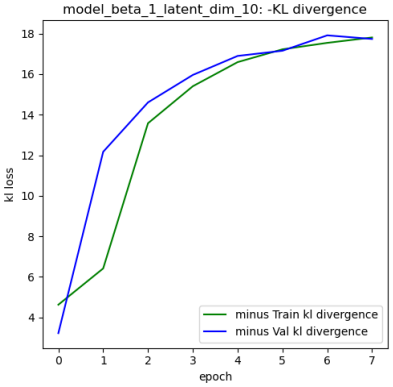




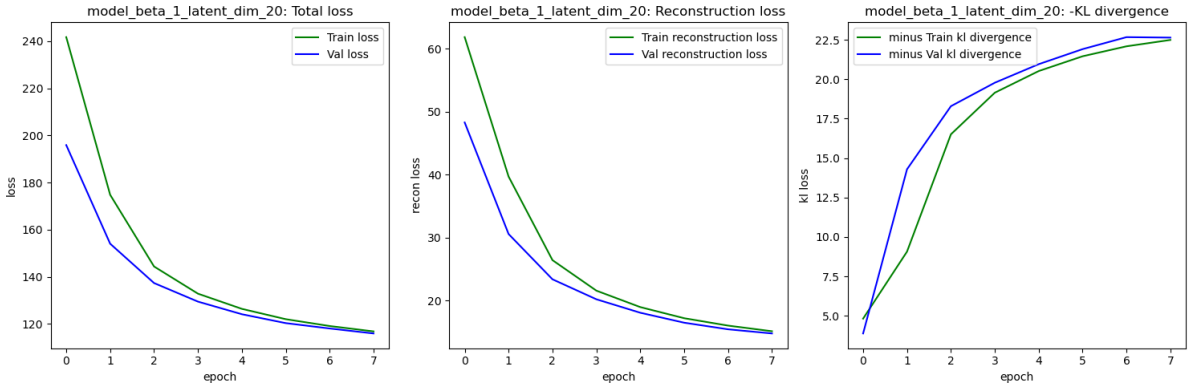
plotting losses for model_beta_1_latent_dim_10







plotting losses for model_beta_1_latent_dim_20



2. Hyper Parameters Tuning

e. Final comparison

The two models that got the best reconstruction error and kl divergence balance are with latent dimension 10. With Beta = 0.75 and Beta = 1 Let's print out their final results to be able to compare the exact numbers

In [324…
```
model_beta_075 = get_model_name(beta=0.75, latent_dim=10)
model_beta_1 = get_model_name(beta=1, latent_dim=10)

print("last epoch results for model with beta = 0.75, latent dimension = 10"
print(additional_final_results_dict[model_beta_075])

print("\nlast epoch results for model with beta = 1, latent dimension = 10")
print(final_results_dict[model_beta_1])
```

```
last epoch results for model with beta = 0.75, latent dimension = 10
{'train_total_loss': 111.8925180568321, 'val_total_loss': 111.4292300347222
2, 'train_recon_loss': 16.17551154162837, 'val_recon_loss': 16.013725911458
334, 'train_kl_divergances': -19.340584218941483, 'val_kl_divergances': -1
9.36729679361979}

last epoch results for model with beta = 1, latent dimension = 10
{'train_total_loss': 115.02845352711397, 'val_total_loss': 114.585799696180
56, 'train_recon_loss': 16.108228302600338, 'val_recon_loss': 15.9650826687
28298, 'train_kl_divergances': -17.807378563974417, 'val_kl_divergances': -
17.73446784125434}
```

3. Train model with selected hyper params

The selected values are: Beta = 1, Latent dimension = 10, since their kl divergence was better, as well as a slightly better reconstruction loss. Now, use the entire train set to train a new VAE model with the selected hyper params.

In [340…
```
# Train selected model
torch.manual_seed(rnd_seed)


chosen_beta = 1
chosen_latent_dim = 10

chosen_model = VAE(chosen_latent_dim).to(device)

chosen_model.train()
optimizer = torch.optim.Adam(chosen_model.parameters(), lr=0.0001)

epochs = 10
```

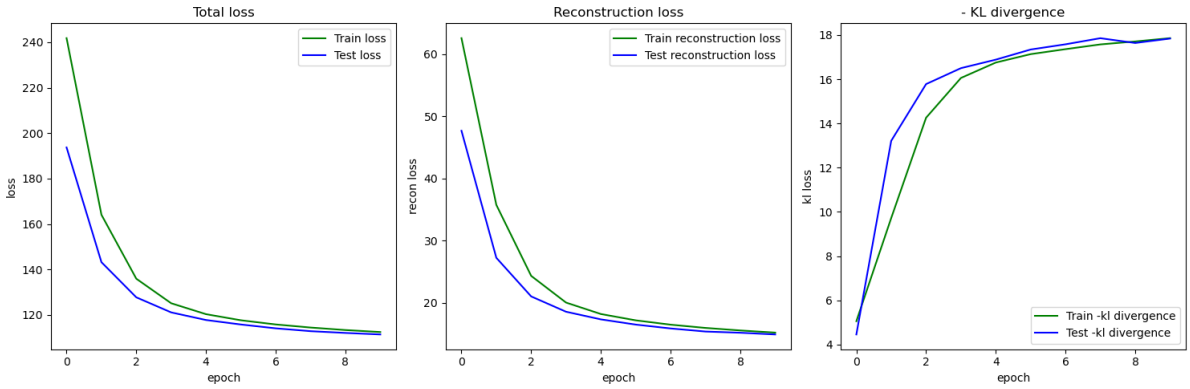
```
train_losses = []
train_recon_losses = []
train_kl_divergances = []

test_losses = []
test_recon_losses = []
test_kl_divergances = []

for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    mean_loss_train, mean_recon_loss_train, mean_kl_divergance_train = trai
    train_losses.append(mean_loss_train)
    train_recon_losses.append(mean_recon_loss_train)
    train_kl_divergances.append(mean_kl_divergance_train)

    mean_loss_test, mean_recon_loss_test, mean_kl_divergance_test = test(te
    test_losses.append(mean_loss_test)
    test_recon_losses.append(mean_recon_loss_test)
    test_kl_divergances.append(mean_kl_divergance_test)
print("Done!")

#########################################################################
#                       ** END OF YOUR CODE **
#########################################################################
```

```
Epoch 1
-------------------------------
training loss: 241.821480
test loss: 193.699822

Epoch 2
-------------------------------
training loss: 164.074250
test loss: 143.195528

Epoch 3
-------------------------------
training loss: 135.921762
test loss: 127.726943

Epoch 4
-------------------------------
training loss: 125.123477
test loss: 121.094731

Epoch 5
-------------------------------
training loss: 120.309106
test loss: 117.738970

Epoch 6
-------------------------------
training loss: 117.628475
test loss: 115.767558

Epoch 7
-------------------------------
training loss: 115.783608
test loss: 114.075113

Epoch 8
-------------------------------
training loss: 114.426248
test loss: 112.827247

Epoch 9
-------------------------------
training loss: 113.323978
test loss: 112.052300

Epoch 10
-------------------------------
training loss: 112.446181
test loss: 111.382986

Done!
```

In [333…

```
# Plotting code

#########################################################################
#                       ** START OF YOUR CODE **
#########################################################################

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# Plot total loss
axes[0].plot(range(epochs), train_losses, color="green", label='Train loss')
axes[0].plot(range(epochs), test_losses, color="blue", label='Test loss')
axes[0].set_title('Total loss')
```

```
axes[0].set_xlabel('epoch')
axes[0].set_ylabel('loss')
axes[0].legend()


# reconstruction loss
axes[1].plot(range(epochs), train_recon_losses, color="green", label='Train
axes[1].plot(range(epochs), test_recon_losses, color="blue", label='Test rec
axes[1].set_title('Reconstruction loss')
axes[1].set_xlabel('epoch')
axes[1].set_ylabel('recon loss')
axes[1].legend()

# kl loss
axes[2].plot(range(epochs), [-1 * kl for kl in train_kl_divergances], color=
axes[2].plot(range(epochs), [-1 * kl for kl in test_kl_divergances], color='
axes[2].set_title('- KL divergence')
axes[2].set_xlabel('epoch')
axes[2].set_ylabel('kl loss')
axes[2].legend()

# Display the plots
plt.tight_layout()
plt.show()



#########################################################################
#                         ** END OF YOUR CODE **
#########################################################################
```



Plot loss (3 points)

Analyze and discuss:

1. Loss curves (reconstruction and KL divergence)
2. Explain how different values of β affect your training.

YOUR ANSWER

1. First, since the kl divergence is multiplied by MINUS β, I decided it to multiply it by minus 1 before plotting, to better demonstrate the contribution it makes to the entire loss function.

   a. -KL divergence graph: The -kl divergence graph begins small, and increases with the epochs. This means at the beginning of the process, before the model learns anything, it is small, and doesn't add to the loss of the model very much. At the

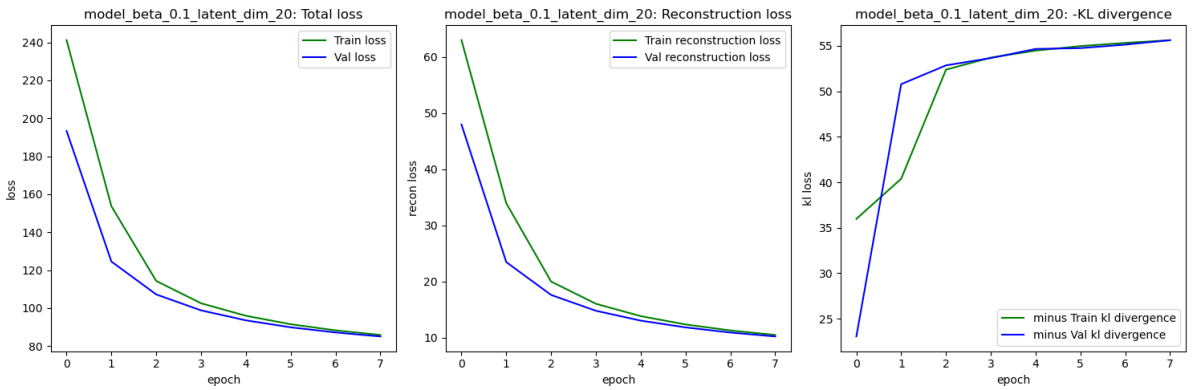
beginning of the training process, the majority of the loss is from the construction loss, since the model still doesn't know how to encode and decode the images. The more the model learns, the larger the regularization term becomes, but the increase becomes smaller in every epoch. This is inline with the kl divergence role and definition: The kl divergence measures how much the latent varibale distribution differs from the standard normal distribution (Normal distribution with mu=0, sigma=1). Therefore, at the beginning when the model doesn't know how to encode the data, there is no reason for the latent variable distribution to differ from the standard normal distribution: It doesn't contain any information anyways. The more the reconstruction error decreases and the model learns, the more information should be encoded into the latent variable. Therefore, it becomes less and less close to the standard normal distribution, and the kl divergence becomes larger. It increases much less in the final epochs, where beta * kl_divergence becomes equal to the reconstruction loss, and it acts as a meaningful regularization component to the model's loss.

b. The reconstruction loss begins very high, as the model doesn't know how to reconstruct an image at the beginning of the process. The more the model learns, the reconstruction loss decreases as the model reconstruct images that are more and more similar to the original input.

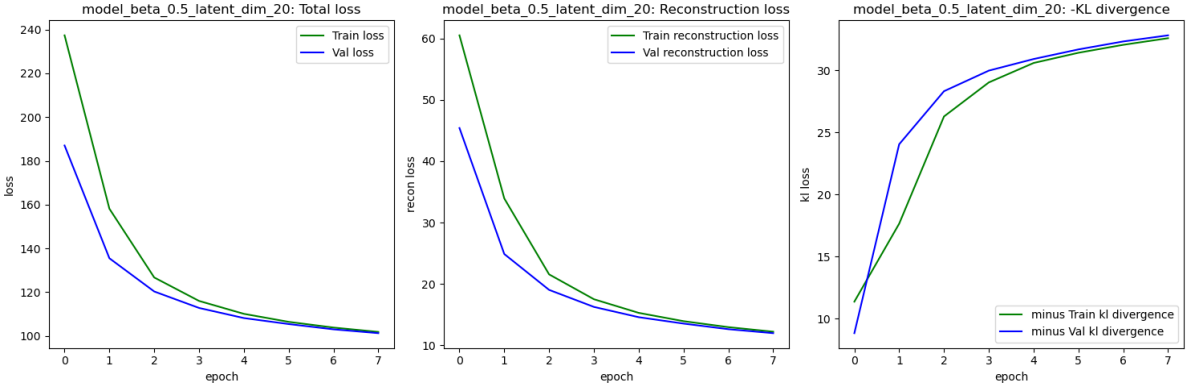
1. The β param dictates how much weight would the training put on the kl divergence. Therefore, Small β values will result in the kl_divergence component being a small part of the general VAE loss, and the reconstruction loss being the main component of the VAE loss. This causes the training process to focus on reducing the reconstruction loss, and putting less emphasis on the disentanglement that's enfoced by the kl divergence component of the loss. In terms of the plots, the smaller the beta value is, the smaller the reconstruction loss will get, as the optimization will focus on improving it over improving the kl divergence. On the other hand, if the β value is larger, the kl divergence component becomes a more meaningful component of the entire loss. Therefore, the learning process will be more focus on improving the kl divergence loss, in the expense of getting higher reconstruction loss. Bellow is a plot to demonstrate this affect.

In [334…
```
## Visualization: how the value of beta affect the training process:
for beta in betas:
    model_name = get_model_name(beta, latent_dim=20)
    plot_model_losses(model_name, full_training_results_dict[model_name],EP(
```

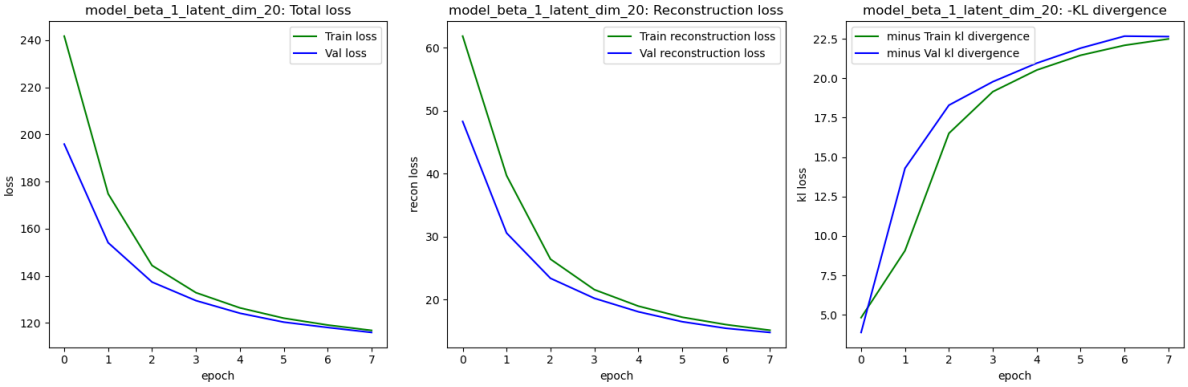
```
plotting losses for model_beta_0.1_latent_dim_20
```



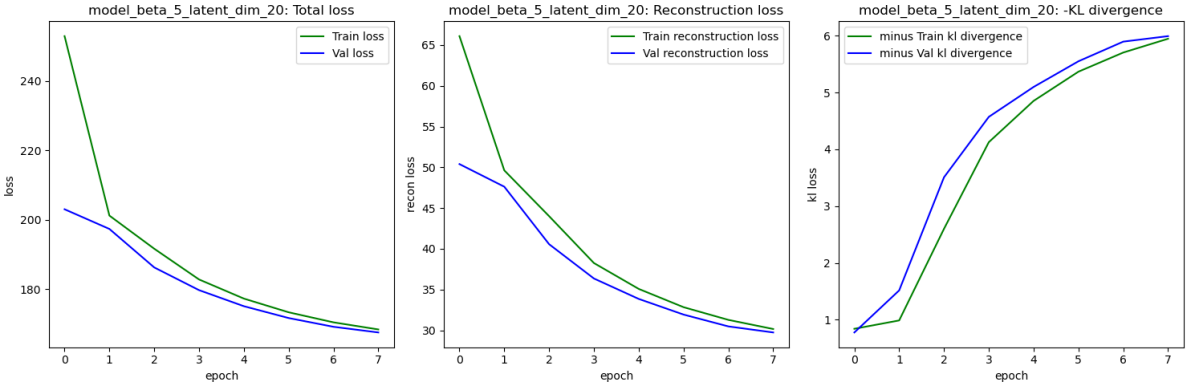
```
plotting losses for model_beta_0.5_latent_dim_20
```




```
plotting losses for model_beta_1_latent_dim_20
```




```
plotting losses for model_beta_5_latent_dim_20
```



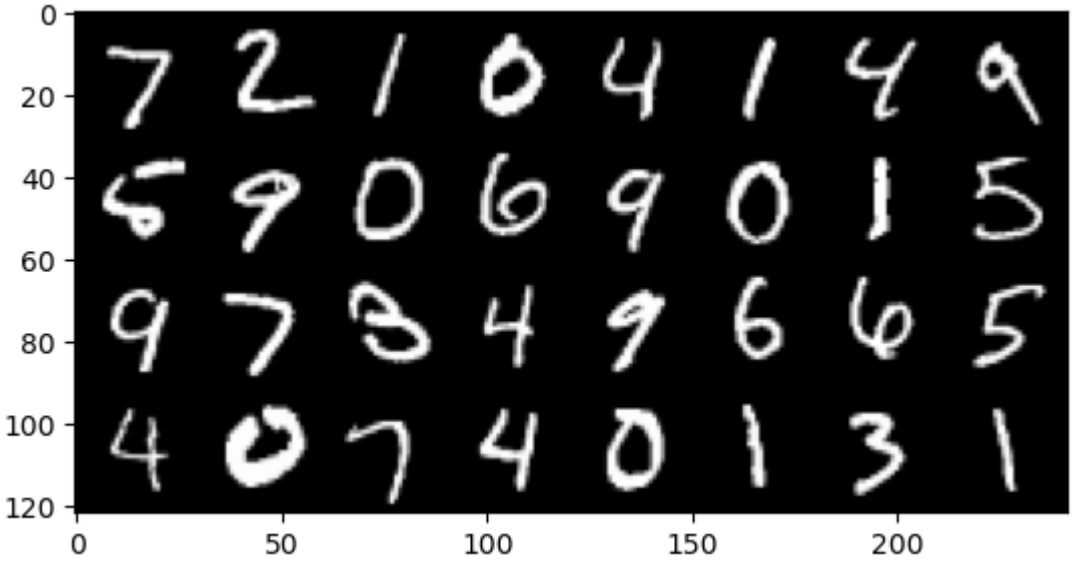
Sample and reconstruction quality (6 points)

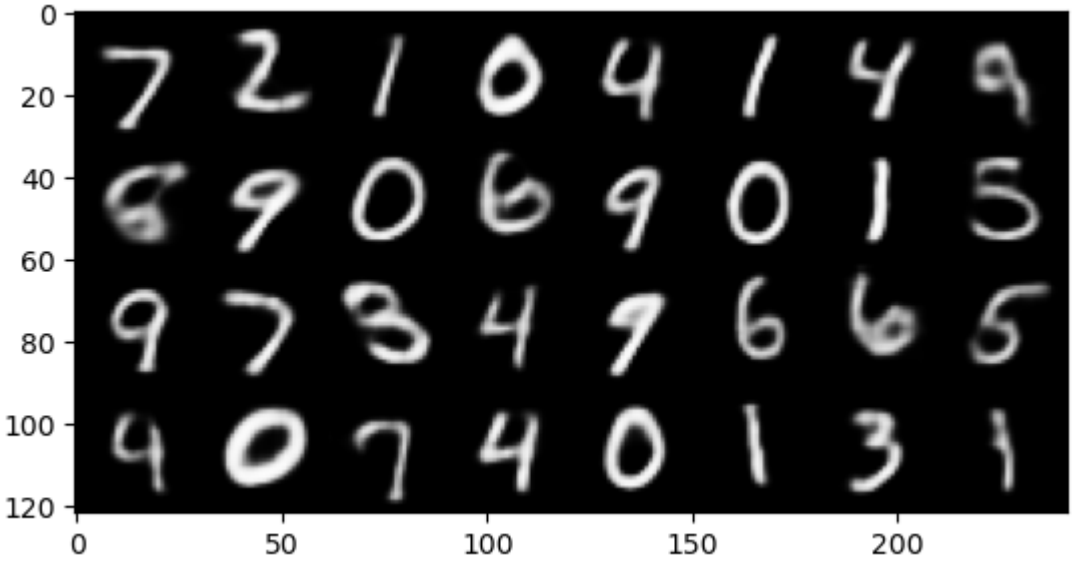
Simply run the below cell to show the output

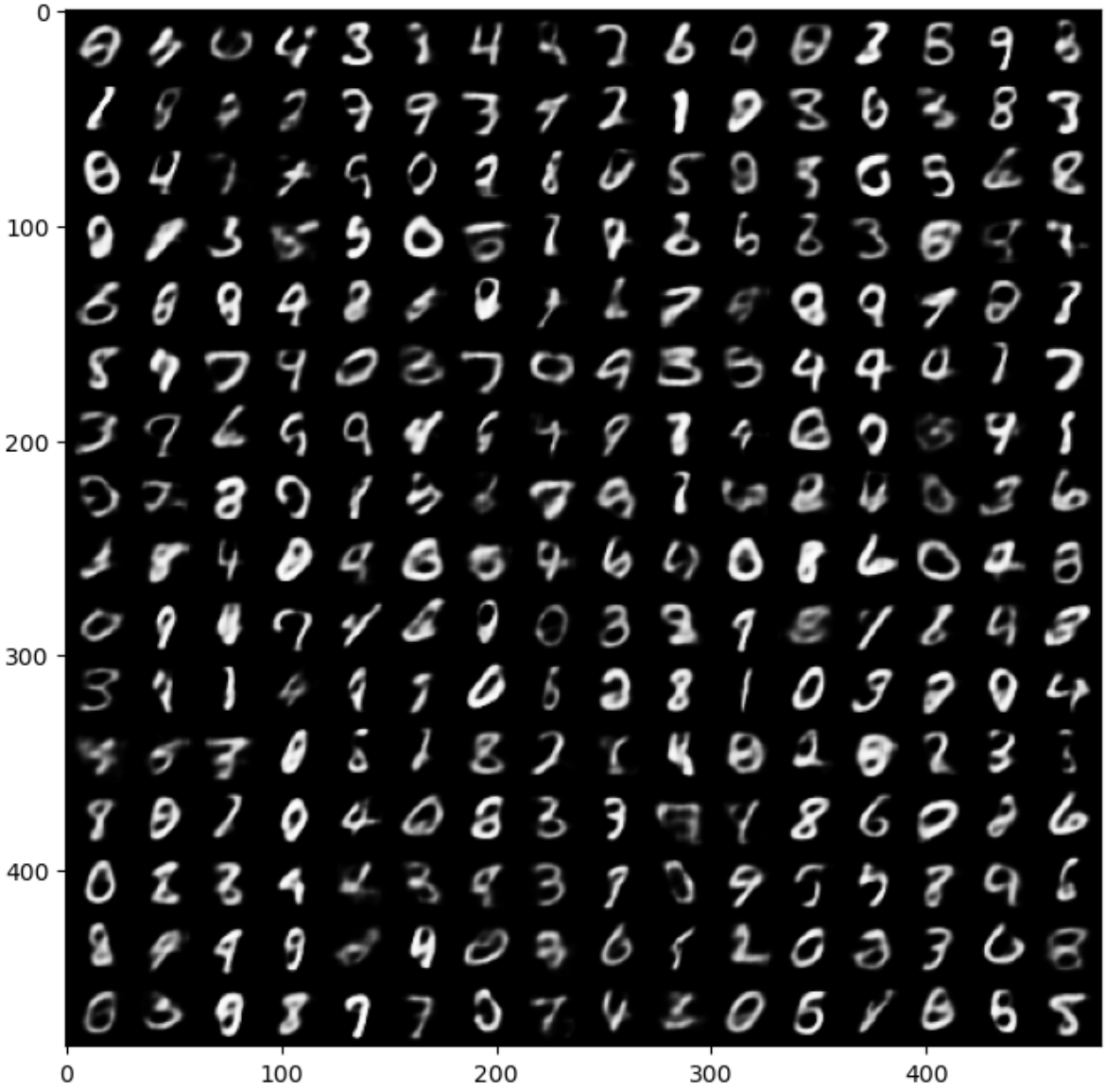
In [367…

```
# Input images
def plot_reconstruction_and_sample(model, latent_dim):
    model.eval()
    sample_inputs, _ = next(iter(test_dataloader))
    fixed_input = sample_inputs[0:32, :, :, :]

    # visualize the original images of the last batch of the test set
    img = make_grid(denormalize(fixed_input), nrow=8, padding=2, normalize=I
                    scale_each=False, pad_value=0)
    plt.figure()
    show(img)

    # Reconstructed images
    with torch.no_grad():

        _, _, recon_batch = model(sample_inputs.to(device))
        recon_batch = recon_batch.unsqueeze(1).reshape(-1,1,28,28)
```

```
        recon_batch = recon_batch[0:32, :, :, :]
        recon_batch = recon_batch.cpu()
        recon_batch = make_grid(denormalize(recon_batch), nrow=8, padding=2,
                                scale_each=False, pad_value=0)
        plt.figure()
        show(recon_batch)

    # Generated Images
    n_samples = 256
    z = torch.randn(n_samples,latent_dim).to(device)
    with torch.no_grad():

        samples = model.decode(z)
        samples = samples.unsqueeze(1).reshape(-1,1,28,28)
        samples = samples.cpu()
        samples = make_grid(denormalize(samples), nrow=16, padding=2, norma
                                scale_each=False, pad_value=0)
        plt.figure(figsize = (8,8))
        show(samples)

plot_reconstruction_and_sample(chosen_model, chosen_latent_dim)
```







Reconstruction and generated samples discussion (5 points)

Analyze and briefly discuss:

1. Reconstruction quality
2. Generated samples quality

Explain in your answers how they relate to different values of β, latent dimension and VAE architecture

YOUR ANSWER

1. Overall, the reconstruction quality is not bad, especially if taking into account the fact that the information was compressed into a 10 dimension vector and then decompressed from it. Almost all digits are recognizable though blurry, and the ones that aren't recognizable are decompressed and reconstructed from images that are not very clear to begin with (The tilted 9 on the first row on the right, the odd 5 on the second row to the left, and the nineish 4 on the bottom row to the left).

- Relation to different values of β: The lower the value of β is, the better the reconstruction will be. This is due to the fact that lower beta means lower importance to the kl divergence component in the loss function, and higher significance to the reconstruction loss in the loss function and thus in the training process. This fact is also demonstrated in the plots bellow, of models with various values of β:
    - The model with Beta = 0.1 reconstructs the digits successfully, with all of them being recognizable and a bit less blurry.
    - The model with Beta = 5 reconstructs blurry images, and even swaps some digits. For exmaple, in the bottom row in the centre it switched 0 into 8 and 4 into 9
- Relation to latent dimension: The latent dimension acts as "the bottleneck" of the model: all the information for the reconstruction should be encoded into this dimension. Therefore, if the dimension is too low, the model might be lacking crusial information required for the reconstruction, and therefore the quality of the reconstructions might be limited. However, if the latent dimension is too high, it might contain unimportant information that will result in unseccessfull reconstruction. For example, I plotted bellow models with the same β as my chosen model (1) but with latent dimensions of 20 and 30. The one with dimension 20 is reconstructing recognizable digits for every digit in the sample. The 30 dimension model, however, changes 5 and 4 into a 9, in the leftmost column

- The architecture of the VAE impacts the reconstruction quality as it determines what relations and patterns the model will be able to extract from the image encode, and how it will be able to extract and decode the latent vector.

1. The quality of the generated samples of my chosen model is more varied: In some cases my model was able to accurately generate a hand writing form of an existing digit. In other cases, it created a grey weak digit, and in some cases it creates a sign that could have been a digit but is not.

    - Relation to different values of β: Generally speaking, higher Beta is correlated with better generated images. This is because higher β values means higher enforcement of the latent space disentangelment by a larger regularization term out of the entire loss function. Disentanglement of the latent space means that every dimension of the latent space is used for representing one specific semantically meaningful feature. For example, in faces images, one dimension of the latent space can encode the angle that the face is looking. Disentangled latent space allows better image generation, due to the fact that the value of every feature can be sample individualy without harming the reliability of the entire image and without impacting the other features. Therefore, when sampling a vector in the latent dimension, we sample the different independent features of the digit, without harming the other features of it. (In the face example, we can sample the face's angle, the hair color, and the shape of the nose, and get a perfectly reliable face)

- Relation to latent dimension: When sampling and generating images for a model with high latent space (30), the generated images are less successful and reliable than the ones for a 10 dimension latent space. In my opinion this could be a result of
    - The larger the dimension is, the harder it is to enforce latent space disentanglement, and therefore the harder it is to successfully sample different features of the digits.
    - Digits are well defined signs, and there are only 10 of them. It is possible that the high dimensionality of the latent space allows sampling of features that are not essential for creating the digits, which adds noise and unrelevant features to the created images.

  On the other hand, it is possible that if the latent dimension is too low, it will not be expressive enough to allow passing enough features for creating a reliable digit. However, I didn't run such a model to validate this idea.

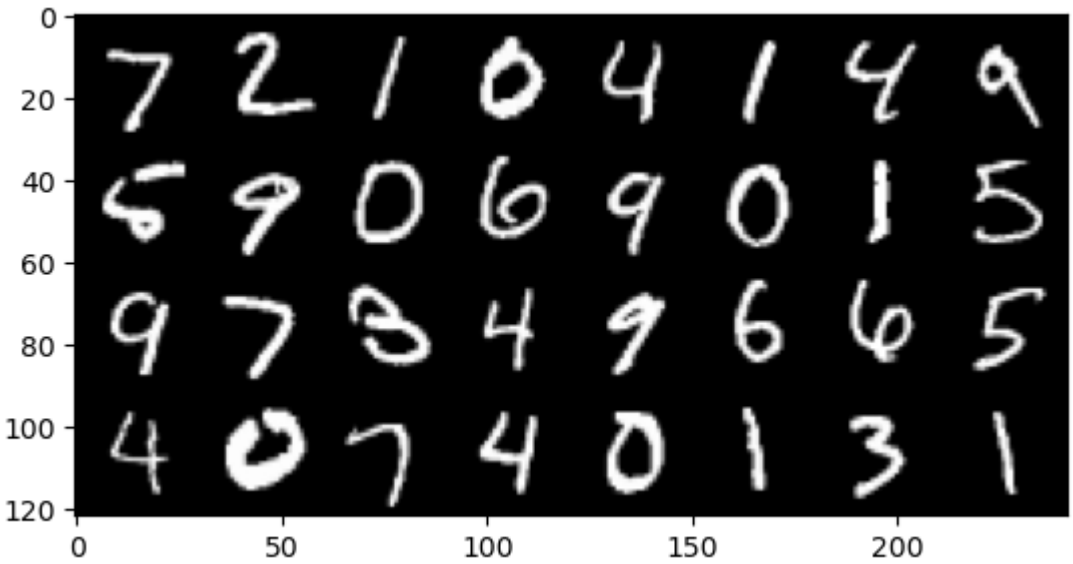
Sample and reconstruction of models with different params

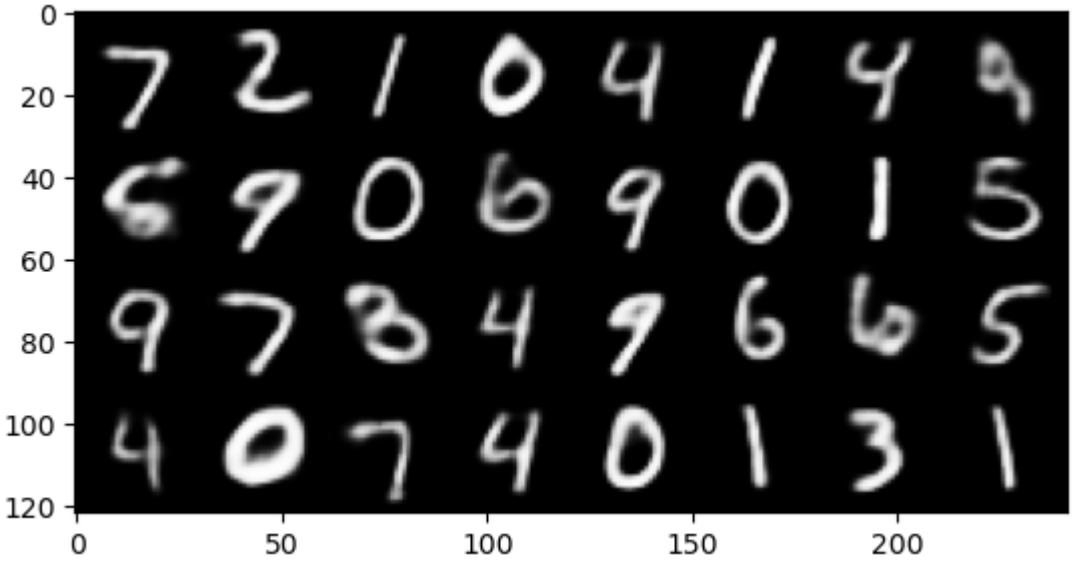
Bellow images are reconstructed and generated by models with different betas and latent dimensions. I used this code and images to investigate and have a more informed discussion in this section.

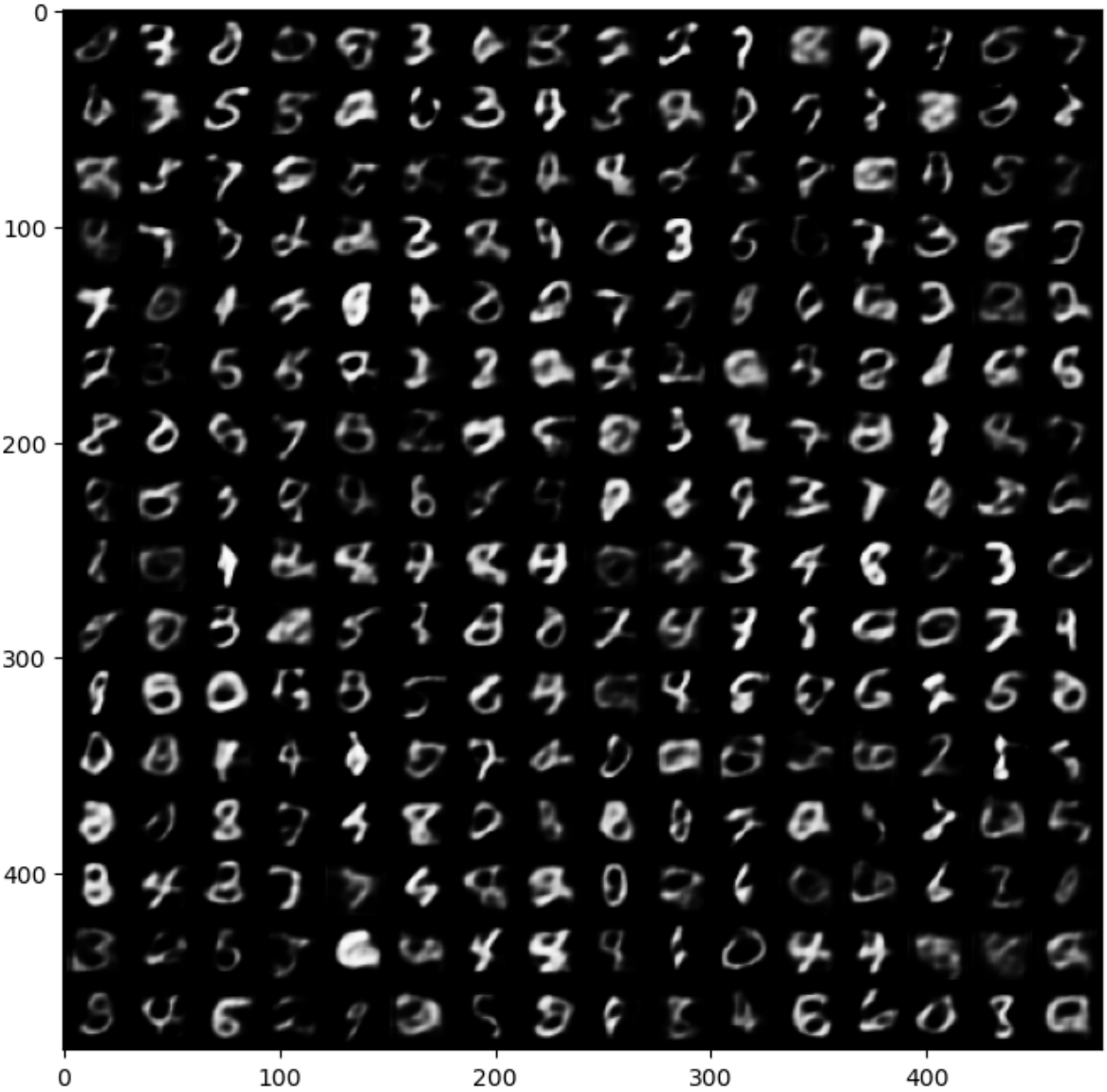
Beta = 0.1

In [370…
```
low_beta_model = pickle.load(open(os.path.join(MODELS_DIR, "model_beta_0.1_
plot_reconstruction_and_sample(low_beta_model, latent_dim=10)
print("Beta = 0.1")
```

```
Beta = 0.1
```





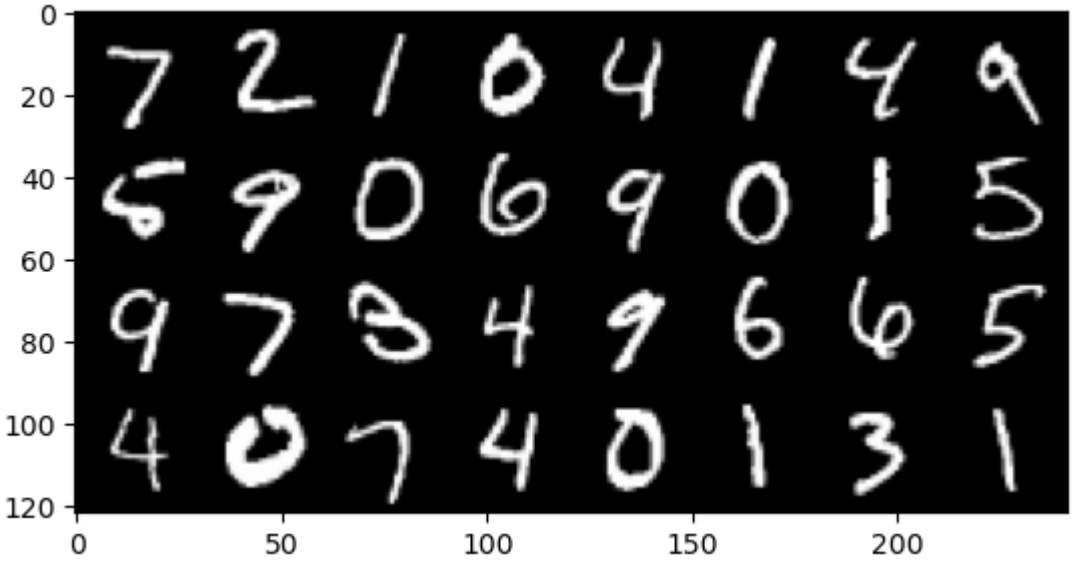


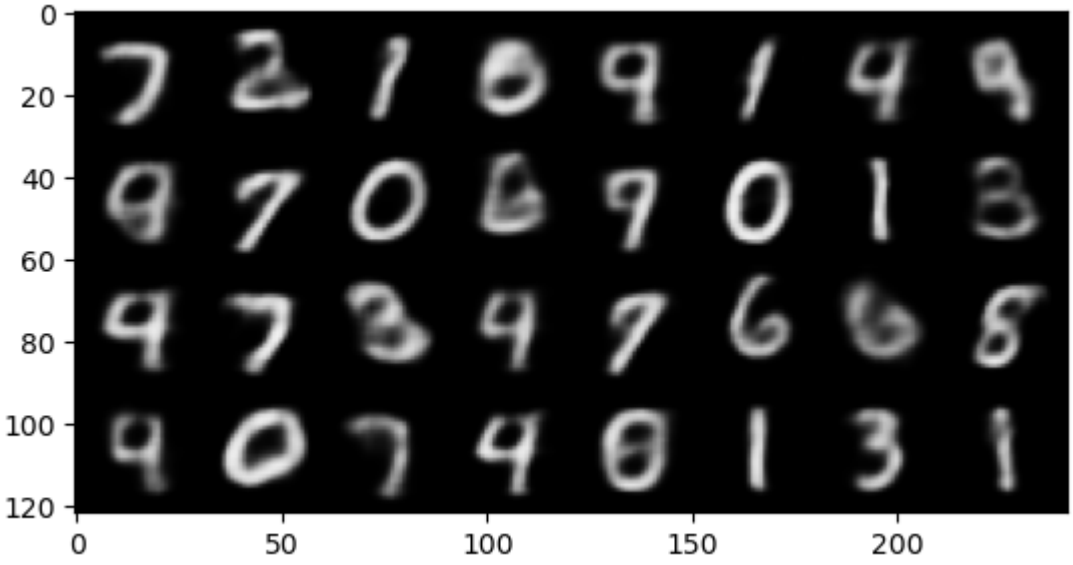
Beta = 5

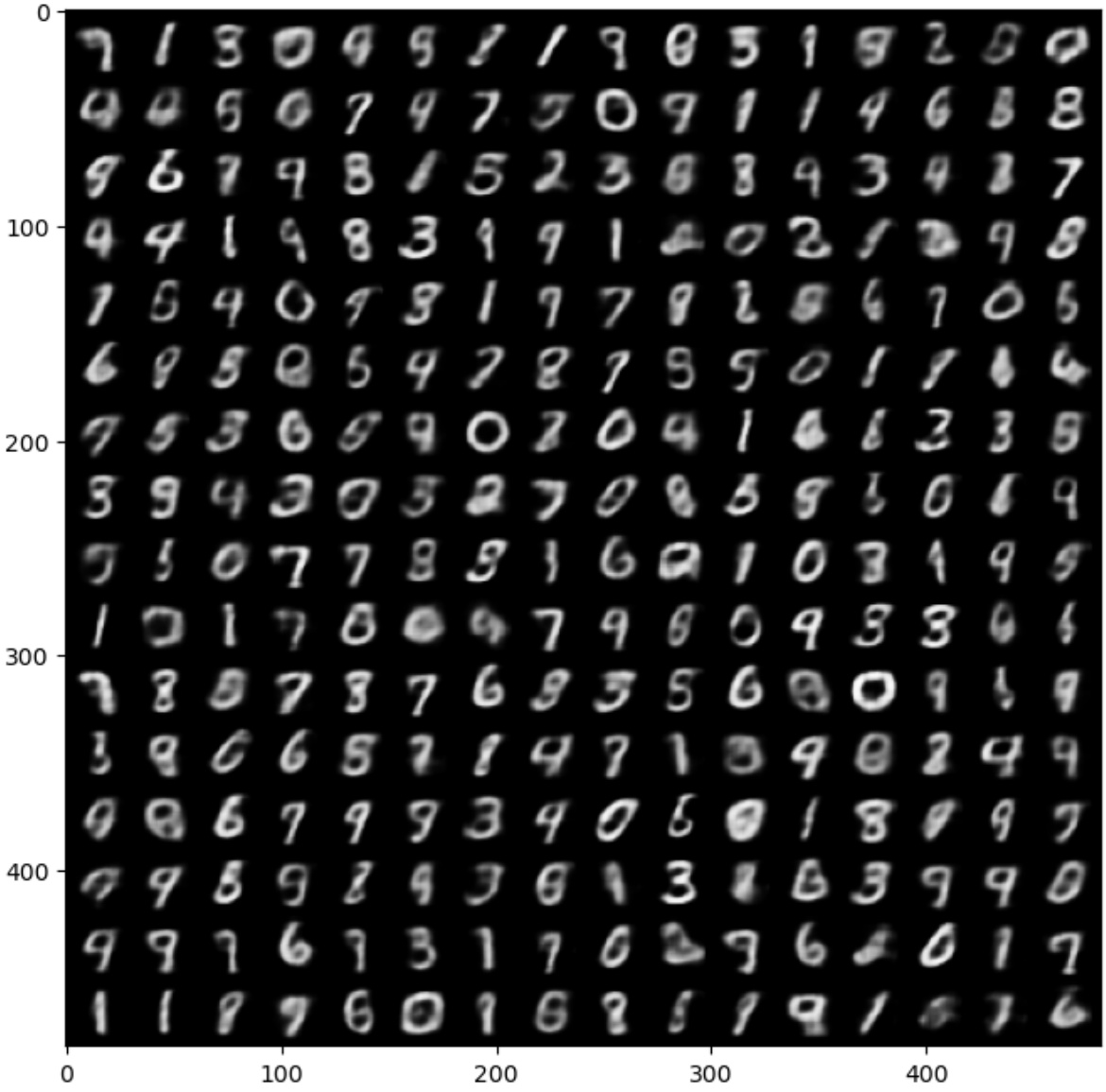
In [361…

```
high_beta_model = pickle.load(open(os.path.join(MODELS_DIR, "model_beta_5_la
plot_reconstruction_and_sample(high_beta_model, latent_dim=10)
```

```
torch.Size([256, 10])
```

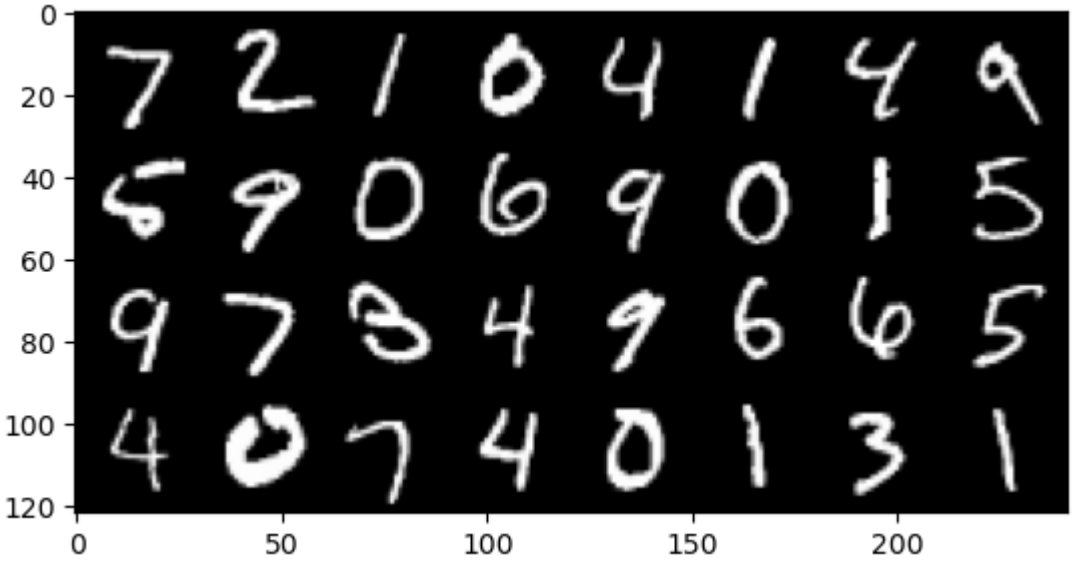

0
20
40
60
80
100
120
0
50
100
150
200

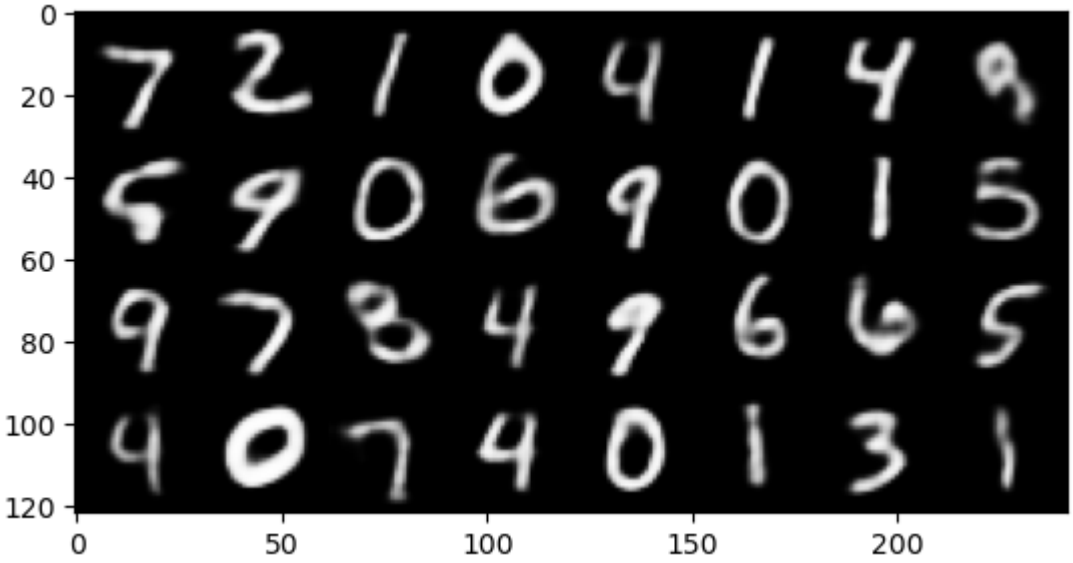

0
20
40
60
80
100
120
0
50
100
150
200

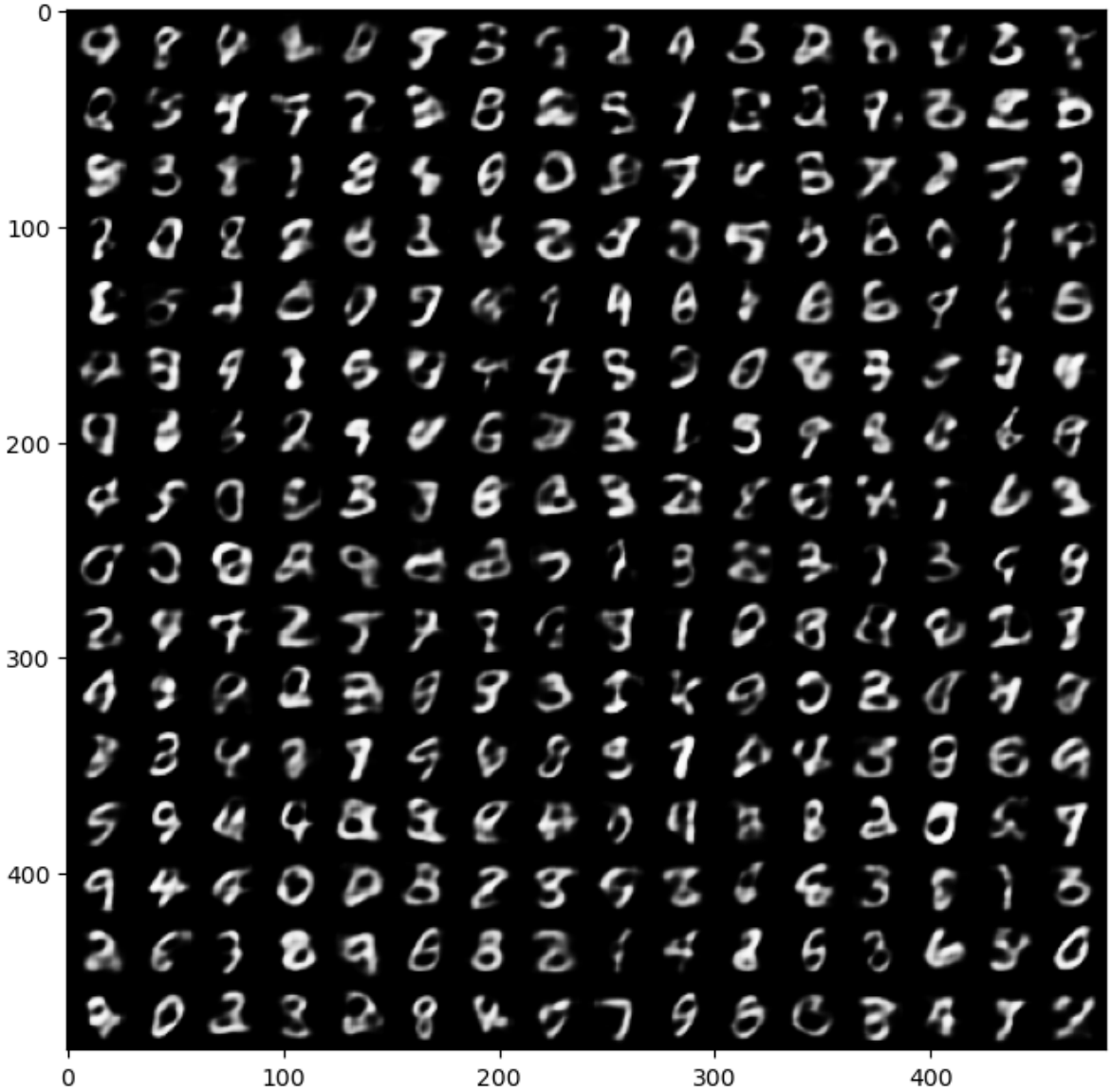


Latent dimension = 20

In [376…
```
middle_dim_model = pickle.load(open(os.path.join(MODELS_DIR, "model_beta_1_
plot_reconstruction_and_sample(middle_dim_model, latent_dim=20)
```



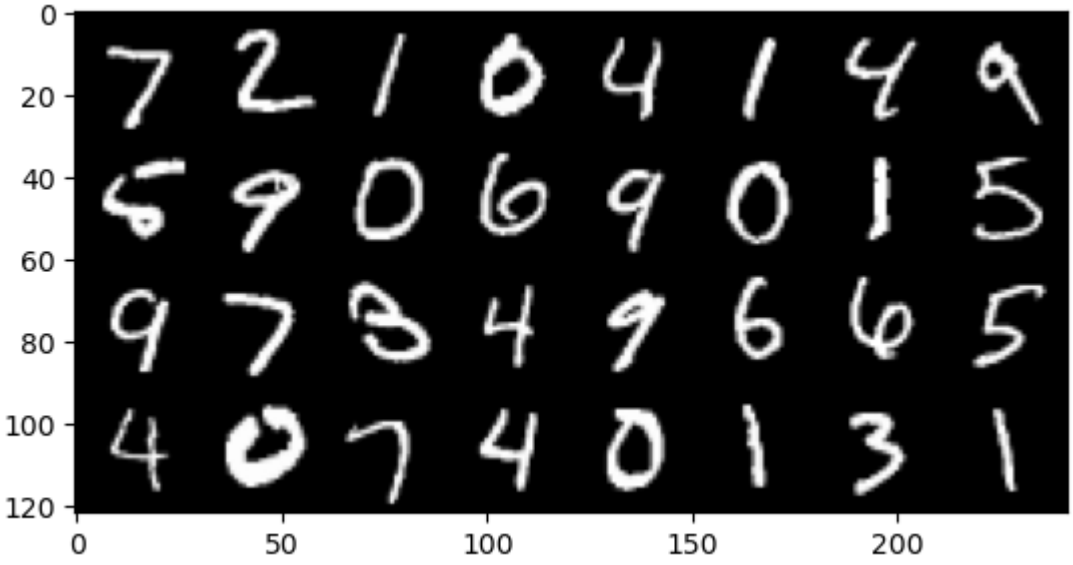


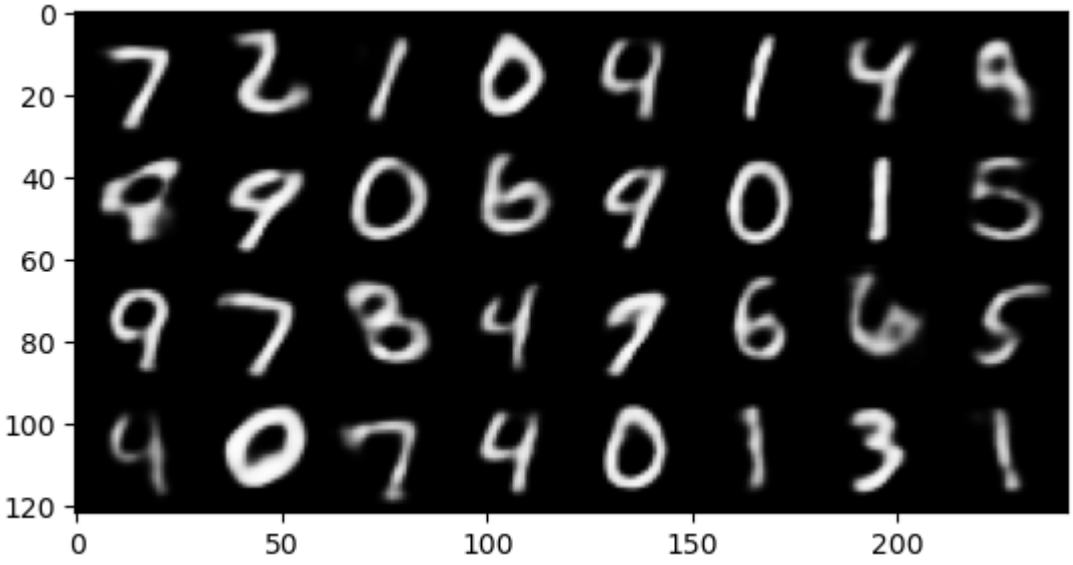


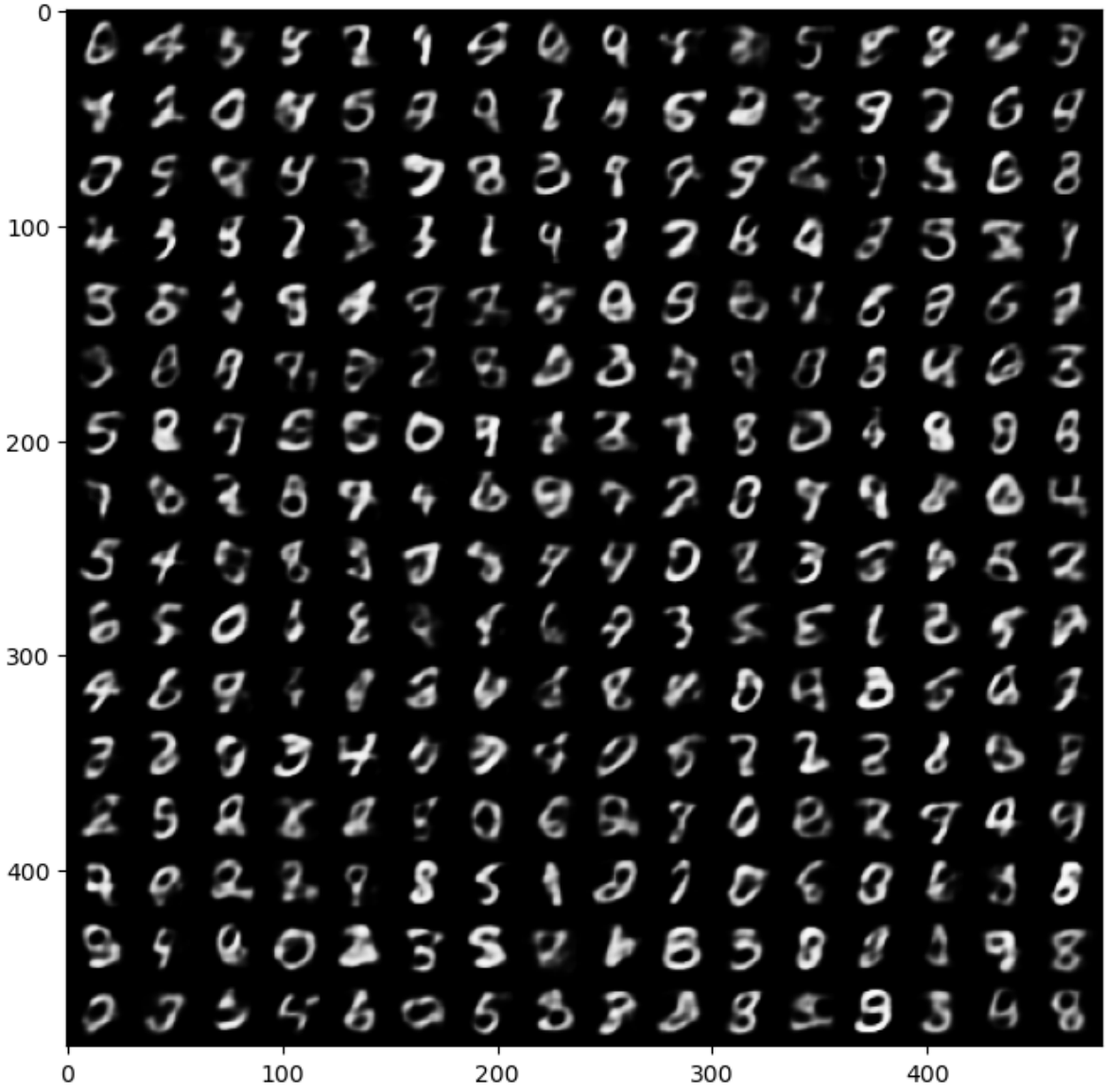
Latent Dimension = 30

In [375…

```
high_dim_model = pickle.load(open(os.path.join(MODELS_DIR, "model_beta_1_lat
plot_reconstruction_and_sample(high_dim_model, latent_dim=30)
```


0
20
40
60
80
100
120
0
50
100
150
200


0
20
40
60
80
100
120
0
50
100
150
200



T-SNE on Embeddings (5 points)

Extract the latent representations of the test set and visualize them using T-SNE

Run the below cells (no coding required).

Qualitatively assess the learned representations of your model using the T-SNE plots.

In [355…

```
alt.data_transformers.disable_max_rows()

def plot_tsne(tsne_xy, dataloader, num_points=1000):

    images, labels = zip(*[(x[0].numpy()[0,:,:,None], x[1]) for x in dataloa

    num_points = min(num_points, len(labels))
    data = pd.DataFrame({'x':tsne_xy[:, 0], 'y':tsne_xy[:, 1], 'label':labe
                         'image': images})
    data = data.sample(n=num_points, replace=False)

    alt.renderers.set_embed_options('light')
    selection = alt.selection_single(on='mouseover', clear='false', nearest=
    scatter = alt.Chart(data).mark_circle().encode(
        alt.X('x:N',axis=None),
        alt.Y('y:N',axis=None),
```

```
        color=alt.condition(selection,
                            alt.value('lightgray'),
                            alt.Color('label:N')),
        size=alt.value(100),
        tooltip='label:N'
    ).add_selection(
        selection
    ).properties(
        width=400,
        height=400
    )

    digit  = alt.Chart(data).transform_filter(
        selection
    ).transform_window(
        index='count()'           # number each of the images
    ).transform_flatten(
        ['image']                 # extract rows from each image
    ).transform_window(
        row='count()',            # number the rows...
        groupby=['index']         # ...within each image
    ).transform_flatten(
        ['image']                 # extract the values from each row
    ).transform_window(
        column='count()',         # number the columns...
        groupby=['index', 'row']  # ...within each row & image
    ).mark_rect(stroke='black',strokeWidth=0).encode(
        alt.X('column:O', axis=None),
        alt.Y('row:O', axis=None),
        alt.Color('image:Q',sort='descending',
            scale=alt.Scale(scheme=alt.SchemeParams('lightgreyteal',
                            extent=[1, 0]),

            ),
            legend=None
        ),
    ).properties(
        width=400,
        height=400,
    )

    return scatter | digit
```

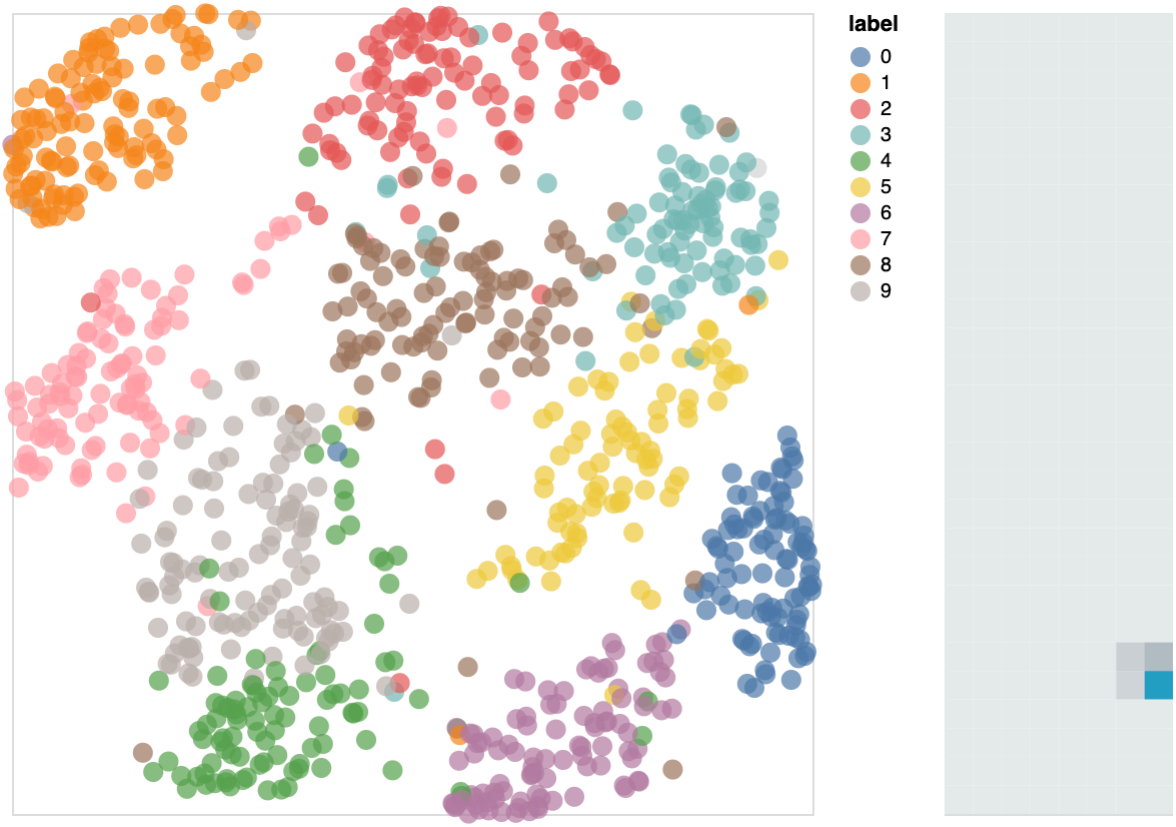
In [356…

```
# TSNE
from sklearn.manifold import TSNE

for t, (x, y) in enumerate(test_dataloader):
    if t == 0:
        data = x
        labels = y
    else:
        data = torch.cat((data, x))
        labels = torch.cat((labels, y))

# Then let's apply dimensionality reduction with the trained encoder

with torch.no_grad():
    data = data.to(device)
    mu, logvar = chosen_model.encode(data)
    z = (chosen_model.reparametrize(mu, logvar)).cpu().detach().numpy()

z_embedded = TSNE(n_components=2).fit_transform(z)
```

In [357…
```
plot_tsne(z_embedded, test_dataloader, num_points=1000)
```

```
/Users/user/anaconda3/lib/python3.11/site-packages/altair/utils/deprecatio
n.py:65: AltairDeprecationWarning: 'selection_single' is deprecated.  Use
'selection_point'
  warnings.warn(message, AltairDeprecationWarning, stacklevel=1)
/Users/user/anaconda3/lib/python3.11/site-packages/altair/utils/deprecatio
n.py:65: AltairDeprecationWarning: 'add_selection' is deprecated. Use 'add_
params' instead.
  warnings.warn(message, AltairDeprecationWarning, stacklevel=1)
```

Out[357]:



Discussion

Analyze and discuss the visualized T-SNE representations

1. What role do the KL loss term and β have?
2. Can you find any outliers?

YOUR ANSWER

The vizualization of the representation of test set samples encoded in the latent space, shows some interesting results: Firstly, we can see that generally speaking, samples with the same labels are clustered together, and located close to one another in the space (this can be easily identified since the point that represent each sample is coloured according to its label).

Secondly, we can see that cluster of closer digits are located close to one another in the space, while clusters that represent digits that are very different from one another are located far in the space. For example: the cluster of the digit 1 (orange) is located at the top left corner, with some distance between its edges and the closest clusters, as no digit is particularly similar to the digit 1. The closest clusters to it are 7 and 2, both share

with the digit 1 the long tilted line in the middle of the image. On the other hand, clusters of digits that are relatively similar to one another are located very close in the space, and might even mix in the edges. For example, the cluster of 4 (green) and 9 (grey) are very close to one another and even mixed in the edges. This is inline with the fact that 4 and 9 tend to be very similar in some hand writings, and share one vertical line and half a circle in the top.

1. The KL loss term's role in regards to the representation of samples in latent space, is to encourage the model to create a representation that is as close as possible to the standard normal distribution. In the 2 dimensional space we use for plotting here, this means the dots will be normally distributed arround the centre of a circle. The role of β in this is to control how much weigh is the kl loss taking out of the entire loss function of the model, and how much emphasis will be put on the regularization in the training process. To demonstrate that, I plotted bellow the representation of latent space for a model with latent dimension = 10 and β = 5. Which means, in the training process of the model there was a big significance to the regularization term. The result shows a distribution that is almost a perfect circle in the middle of the space, that is not well seperated into the different clusters. In my model's case, β = 1, so there is a balance between a circle like shape of the latent space representation, to still a clear separation into separated clusters between the different labels.

1. There are a few outliers in the representation of the test samples in the latent space. I detailed about a few of them and tried to reason about why they were located far from their cluster in the latent space:
    - In the cluster of 1 at the top left corner, there is one 9 digit. Its location in the space could be explained by its tiny circle at the top, which is different from many 9 digits that have a more significant circle. This makes this 9 look similar to 1, as its main feature is the line along the image, and confuses the model.
    - In the side of the 2 cluster, there is one 4 digit, which has a large circle on the left bottom, similarly to the digit 2.
    - In the middle of the 9 cluster, there are 2 samples of the digit 4. Both of them has a very thin open in the top, which makes their top form almost a close shape, just like the circle on the top of the digit 9.

T-SNE embeddings of representation by model with β=5

To support arguments in the discussion above

In [378…

```
# TSNE
from sklearn.manifold import TSNE

for t, (x, y) in enumerate(test_dataloader):
    if t == 0:
        data = x
        labels = y
    else:
        data = torch.cat((data, x))
        labels = torch.cat((labels, y))

# Then let's apply dimensionality reduction with the trained encoder
```

```
with torch.no_grad():
    data = data.to(device)
    mu, logvar = high_beta_model.encode(data)
    z = (high_beta_model.reparametrize(mu, logvar)).cpu().detach().numpy()

z_embedded = TSNE(n_components=2).fit_transform(z)

plot_tsne(z_embedded, test_dataloader, num_points=1000)
```

```
/Users/user/anaconda3/lib/python3.11/site-packages/altair/utils/deprecatio
n.py:65: AltairDeprecationWarning: 'selection_single' is deprecated.  Use
'selection_point'
  warnings.warn(message, AltairDeprecationWarning, stacklevel=1)
/Users/user/anaconda3/lib/python3.11/site-packages/altair/utils/deprecatio
n.py:65: AltairDeprecationWarning: 'add_selection' is deprecated. Use 'add_
params' instead.
  warnings.warn(message, AltairDeprecationWarning, stacklevel=1)
```

Out[378]: