DNN Speech Enhancement Tutorial

In the jupyter notebook, we will use pytorch to implement a simple DNN that used for speech enhancement.

1. Import package

First, we will import all packages we need for this project:

In [1]:
```
import librosa
# import ffmpeg
import os
import torch
import torch.nn.functional as Func
import torch.nn as nn
import numpy as np
from torch.autograd import Variable
import torch.optim as optim
import torch.utils.data as data
from torch.nn.utils.rnn import pack_padded_sequence
from torch.nn.utils.rnn import pad_packed_sequence
import copy
train_noisyPath = 'PREPARED_DATASET/TRAIN/'  # Used
train_cleanPath = 'PREPARED_DATASET/CLEAN/TRAIN/'
dev_noisyPath = 'PREPARED_DATASET/DEV/'  # Used
dev_cleanPath = 'PREPARED_DATASET/CLEAN/DEV/'
test_noisyPath = 'PREPARED_DATASET/TEST/'
test_cleanPath = 'PREPARED_DATASET/CLEAN/TEST/'
restfiles_Path = 'PREPARED_DATASET/REST_FILES/'

import os

train_clean_male_filenames=os.listdir(r"Speech Data/IEEE/IEEE_male/train_male")
dev_clean_male_filenames=os.listdir(r"Speech Data/IEEE/IEEE_male/development_mal
test_clean_male_filenames=os.listdir(r"Speech Data/IEEE/IEEE_male/test_male")

train_clean_female_filenames=os.listdir(r"Speech Data/IEEE/IEEE_female/train_fema
dev_clean_female_filenames=os.listdir(r"Speech Data/IEEE/IEEE_female/development_
test_clean_female_filenames=os.listdir(r"Speech Data/IEEE/IEEE_female/test_femal

print("CLEAN--> Male: Train Length: {} , Dev Length {} , Test length {} ".format
print("CLEAN--> Female: Train Length: {} , Dev Length {} , Test length {} ".form

train_noisy_male_filenames=os.listdir(r"PREPARED_DATASET/TRAIN_MALE/")
dev_noisy_male_filenames=os.listdir(r"PREPARED_DATASET/DEV_MALE/")
test_noisy_male_filenames=os.listdir(r"PREPARED_DATASET/TEST_MALE/")

train_noisy_female_filenames=os.listdir(r"PREPARED_DATASET/TRAIN_FEMALE/")
dev_noisy_female_filenames=os.listdir(r"PREPARED_DATASET/DEV_FEMALE/")
test_noisy_female_filenames=os.listdir(r"PREPARED_DATASET/TEST_FEMALE/")

print("NOISY --> Male: Train Length: {} , Dev Length {} , Test length {} ".forma
print("NOISY --> Female: Train Length: {} , Dev Length {} , Test length {} ".for
################################################################################
################################################################################
# Train Clean Speech
train_cleanSpeechList = train_clean_male_filenames+train_clean_female_filenames
train_cleanSpeechLength = len(train_cleanSpeechList)
print("Train Clean Total length (Male + Female) : ",train_cleanSpeechLength)

#.npy output folder
train_clean_PyPath = './Data/npy/Train_frame/Clean'

# Train Noisy Speech
train_noisySpeechList = train_noisy_male_filenames+train_noisy_female_filenames
```

```
train_noisySpeechLength = len(train_noisySpeechList)
print("Train Noisy Total length (Male + Female) : ",train_noisySpeechLength)

#.npy output folder
train_noisy_PyPath = './Data/npy/Train_frame/Noisy'


##########################################################################
##########################################################################
# Dev Clean Speech
dev_cleanSpeechList = dev_clean_male_filenames+dev_clean_female_filenames
dev_cleanSpeechLength = len(dev_cleanSpeechList)
print("Dev Clean Total length (Male + Female) : ",dev_cleanSpeechLength)

#.npy output folder
dev_clean_PyPath = './Data/npy/Dev_frame/Clean'

# Dev Noisy Speech
dev_noisySpeechList = dev_noisy_male_filenames+dev_noisy_female_filenames
dev_noisySpeechLength = len(dev_noisySpeechList)
print("Dev Noisy Total length (Male + Female) : ",dev_noisySpeechLength)

#.npy output folder
dev_noisy_PyPath = './Data/npy/Dev_frame/Noisy'


##########################################################################
# Test Clean Speech
test_cleanSpeechList = test_clean_male_filenames+test_clean_female_filenames
test_cleanSpeechLength = len(test_cleanSpeechList)
print("Test Clean Total length (Male + Female) : ",test_cleanSpeechLength)


test_clean_PyPath = './Data/npy/Test_frame/Clean'

# Test Noisy Speech
test_noisySpeechList = test_noisy_male_filenames+test_noisy_female_filenames
test_noisySpeechLength = len(test_noisySpeechList)
print("Test Noisy Total length (Male + Female) : ",test_noisySpeechLength)

#.npy output folder
test_noisy_PyPath = './Data/npy/Test_frame/Noisy'

##########################################################################
#Data Path for training data
train_noisyPath = 'PREPARED_DATASET/TRAIN/'  # Used
train_cleanPath = 'PREPARED_DATASET/CLEAN/TRAIN/'
dev_noisyPath = 'PREPARED_DATASET/DEV/'  # Used
dev_cleanPath = 'PREPARED_DATASET/CLEAN/DEV/'
test_noisyPath = 'PREPARED_DATASET/TEST/'
test_cleanPath = 'PREPARED_DATASET/CLEAN/TEST/'
restfiles_Path = 'PREPARED_DATASET/REST_FILES/'
```

```
CLEAN--> Male: Train Length: 500 , Dev Length 100 , Test length 100
CLEAN--> Female: Train Length: 500 , Dev Length 100 , Test length 100
NOISY --> Male: Train Length: 4500 , Dev Length 900 , Test length 900
```

```
NOISY --> Female: Train Length: 4500 , Dev Length 900 , Test length 900
Train Clean Total length (Male + Female) :  1000
Train Noisy Total length (Male + Female) :  9000
Dev Clean Total length (Male + Female) :  200
Dev Noisy Total length (Male + Female) :  1800
Test Clean Total length (Male + Female) :  200
Test Noisy Total length (Male + Female) :  1800
```

Q1 Write separate functions to compute the fast Fourier Transform (FFT) mask, ideal binary mask (IBM) and ideal ratio mask (IRM) given the speech and noise segments of a noisy speech signal. For the FFT-mask, truncate the label to values between 0 and 1 (inclusively).

In [2]:
```
def snr_calculate(speech_data,noise_data):
    speech_energy=np.sum(np.array(speech_data, dtype='int64')**2)
    noise_energy=np.sum(np.array(noise_data, dtype='int64')**2)
    ratio=speech_energy/noise_energy
    sound_level=10*math.log(ratio,10)
    return sound_level
def IBM(noisy_speech,clean_speech):
    noise=noisy_speech-clean_speech
    mask=clean_speech
    mask[clean_speech>=noise]=1
    mask[clean_speech<noise]=0
    return mask
def IRM(noisy_speech,clean_speech):
    noise=noisy_speech-clean_speech
    speech_energy=np.array(clean_speech)**2
    noise=np.array(noise)**2
    irm = np.sqrt(speech_energy / (noise + speech_energy))
    return irm
def FFT_mask(noisy_speech,clean_speech):
    # For the FFT-mask, truncate the label to values between 0 and 1 (inclusivel
    return np.clip(clean_speech/noisy_speech,0,1)
```

In [ ]:
```
# Generating absolute value of whole training data
train_noisy_signal=np.load('Train_noisy.npy')  # this is 10*log10(np.abs(NOISYSP
train_noisy_signal=np.power(10,train_noisy_signal/10)
np.save('Abs_Train_noisy.npy',train_noisy_signal) # this is only np.abs(NOISYSPE

# Generating absolute value of whole training data
dev_noisy_signal=np.load('Dev_noisy.npy')  # this is 10*log10(np.abs(NOISYSPEECH
train_noisy_signal=np.power(10,dev_noisy_signal/10)
np.save('Abs_Dev_noisy.npy',train_noisy_signal) # this is only np.abs(NOISYSPEEC
```

In [ ]:
```
noisy_speech=np.load("Abs_Train_noisy.npy")
clean_speech=np.load("
                     .npy")
X_ibm=IBM(noisy_speech,clean_speech)
np.save('train_label_ibm.npy',X_ibm)
del X_ibm
X_irm=IRM(noisy_speech,clean_speech)
np.save('train_label_irm.npy',X_irm)
del X_irm
X_fft=FFT_mask(noisy_speech,clean_speech)
np.save('train_label_fft.npy',X_fft)
del X_fft
noisy_speech=np.load("Abs_Dev_noisy.npy")
clean_speech=np.load("Dev_clean.npy")
X_ibm=IBM(noisy_speech,clean_speech)
np.save('dev_label_ibm.npy',X_ibm)
del X_ibm
X_irm=IRM(noisy_speech,clean_speech)
np.save('dev_label_irm.npy',X_irm)
del X_irm
X_fft=FFT_mask(noisy_speech,clean_speech)
np.save('dev_label_fft.npy',X_fft)
del X_fft
```

Question 2: Using the above functions, train separate DNNs that individually estimate the clean speech spectrogram, FFT mask, IBM, and IRM. In other words, you should have four different DNNs, one for each training target. Use the parameters and network structure that is outlined in the paper. Note that example DNN code is included in the homework folder. Be sure to test the network with the validation/development data after each Epoch and perform model selection with the best performing result. For each DNN, generate and plot error curves (MSE) as a function of Epoch for the training and validation/development set.

AND Question 3: After the networks are trained, test each of the networks with the testing data set. Generate plots that show the average MSE between the estimated and true clean speech time-domain signals for each training target.

Model

Model Structure

Next, we will start to construct our model. In this tutorial, we only use a simple 4-layer DNN.

The input data has a dimension of n * 257 which n is the time stamps and 257 is the number of frequency bins. The dimension of hidden layers and output layer is describe as following. Each layer will be followed by a RELU activation layer.

NORMALIZATION

In [3]:
```
#Dataloader for traning data

# SPECTOGRAM OUTPUT

class trainDataLoader(data.Dataset):
    def __init__(self,label_file,data_file):
        print(label_file,data_file)
        self.labelPath = np.load(label_file)
        self.dataPath = np.load(data_file)
    def __getitem__(self, index):
        xFile = self.dataPath.T[index]
        mFile = self.labelPath.T[index]
        return torch.from_numpy(xFile), torch.from_numpy(mFile)
    def __len__(self):
        #Number of files
        return self.dataPath.shape[1]

class valDataLoader(data.Dataset):
    def __init__(self,label_file,data_file):
        print(label_file,data_file)
        self.labelPath = np.load(label_file)
        self.dataPath = np.load(data_file)
    def __getitem__(self, index):
        xFile = self.dataPath.T[index]
        mFile = self.labelPath.T[index]
        return torch.from_numpy(xFile), torch.from_numpy(mFile)
    def __len__(self):
        return self.dataPath.shape[1]

class testDataLoader(data.Dataset):
    def __init__(self,label_file,data_file):
        self.labelPath = np.load(label_file)
        self.dataPath = np.load(data_file)
    def __getitem__(self, index):
        xFile = self.dataPath.T[index]
        mFile = self.labelPath.T[index]
        return torch.from_numpy(xFile), torch.from_numpy(mFile)
    def __len__(self):
        return self.dataPath.shape[1]
```

In [4]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(257,1024)
        self.fc2 = nn.Linear(1024,1024)
        self.fc3 = nn.Linear(1024,1024)
        self.fc4 = nn.Linear(1024,257)
    def forward(self,audio):
        audio = Func.relu(self.fc1(audio))
        audio = Func.relu(self.fc2(audio))
        audio = Func.relu(self.fc3(audio))
        audio = self.fc4(audio)
        return audio
def weights(m):
    if isinstance(m,nn.Linear):
        nn.init.xavier_normal(m.weight.data)
        nn.init.constant(m.bias.data,0.1)
```

Traning Process

The big for loop for your training. Use everything defined preiously, set number of epoches and train your model.

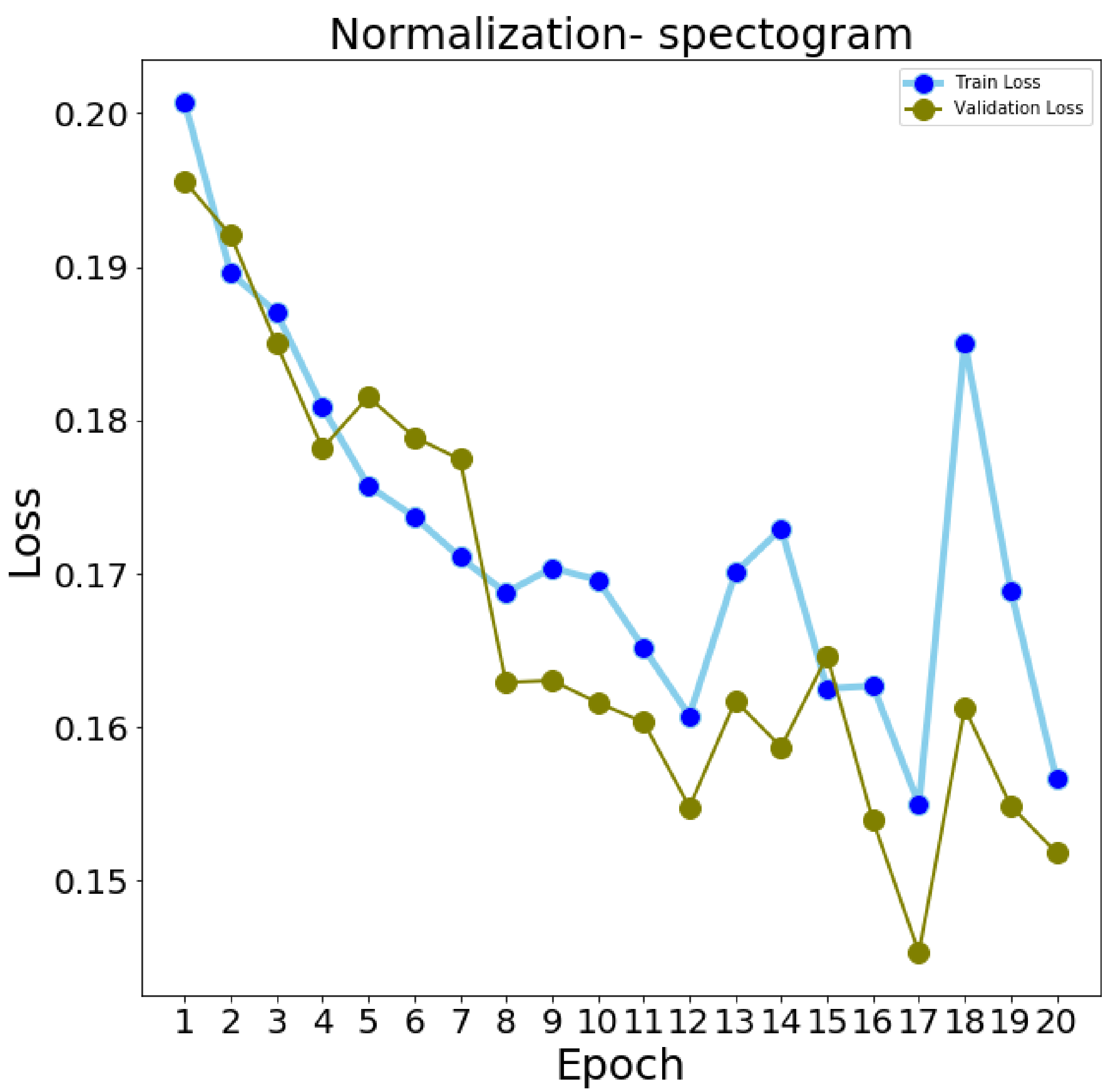
In [5]:
```
def train_model(model,trainData,valData,num_epochs):
    best_model = copy.deepcopy(model.state_dict())
    best_loss = 9999
    train_loss=[]
    validation_loss=[]
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        loss = 0.0
        vali_loss = 0.0
        for step, (audio, target) in enumerate(trainData):
            if(step%15==0):
                print("Train step:"+str(step)+"/"+str(len(trainData)))
            audio=audio.float()
            target=target.float()
            model.train()
            model=model.float()
            output = model(audio)
            newLoss = criterion(output,target)
            loss += newLoss.data
            optimizer.zero_grad()
            newLoss.backward()
            optimizer.step()
        for step, (audio, target) in enumerate(valData):
            audio=audio.float()
            target=target.float()
            model.eval()
            output = model(audio)
            new_valiLoss = criterion(output,target)
            vali_loss += new_valiLoss.data
            if vali_loss < best_loss:
                best_loss = vali_loss
                best_model = copy.deepcopy(model.state_dict())
            #if(step%30==0):
             #   print("Valid step:"+str(step)+"/"+str(len(valData)))
        print('Epoch:{:2}, Train Loss: {:>.5f}'.format(epoch,loss/len(trainData)
        print('Epoch:{:2}, Valid Loss: {:>.5f}'.format(epoch,vali_loss/len(valDa
        train_loss.append(loss/len(trainData))
        validation_loss.append(vali_loss/len(valData))
    return train_loss,validation_loss,best_model
```

In [31]:
```
# NORMALIZATION  SPECTOGRAM

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='Train_clean.npy'
data_file='Normalization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 40
label_file='Dev_clean.npy'
data_file='Normalization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000
train_loss,validation_loss,best_model=train_model(model,trainData,valData,20)
```

```
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:14, Train Loss: 0.16252
Epoch:14, Valid Loss: 0.16465
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.16269
Epoch:15, Valid Loss: 0.15392
Epoch 16/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
```

In [83]:
```
train_loss=np.load("train_loss.npy")
validation_loss=np.load("validation_loss.npy")
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss, marker='o', markerfacecolor='blue', markersize=
plt.plot(range(1,21),validation_loss, marker='o', color='olive',markersize=12, l
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Normalization- spectogram",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
```

Out[83]:
```
Text(0, 0.5, 'Loss')
```



Save your model

You can save your best model

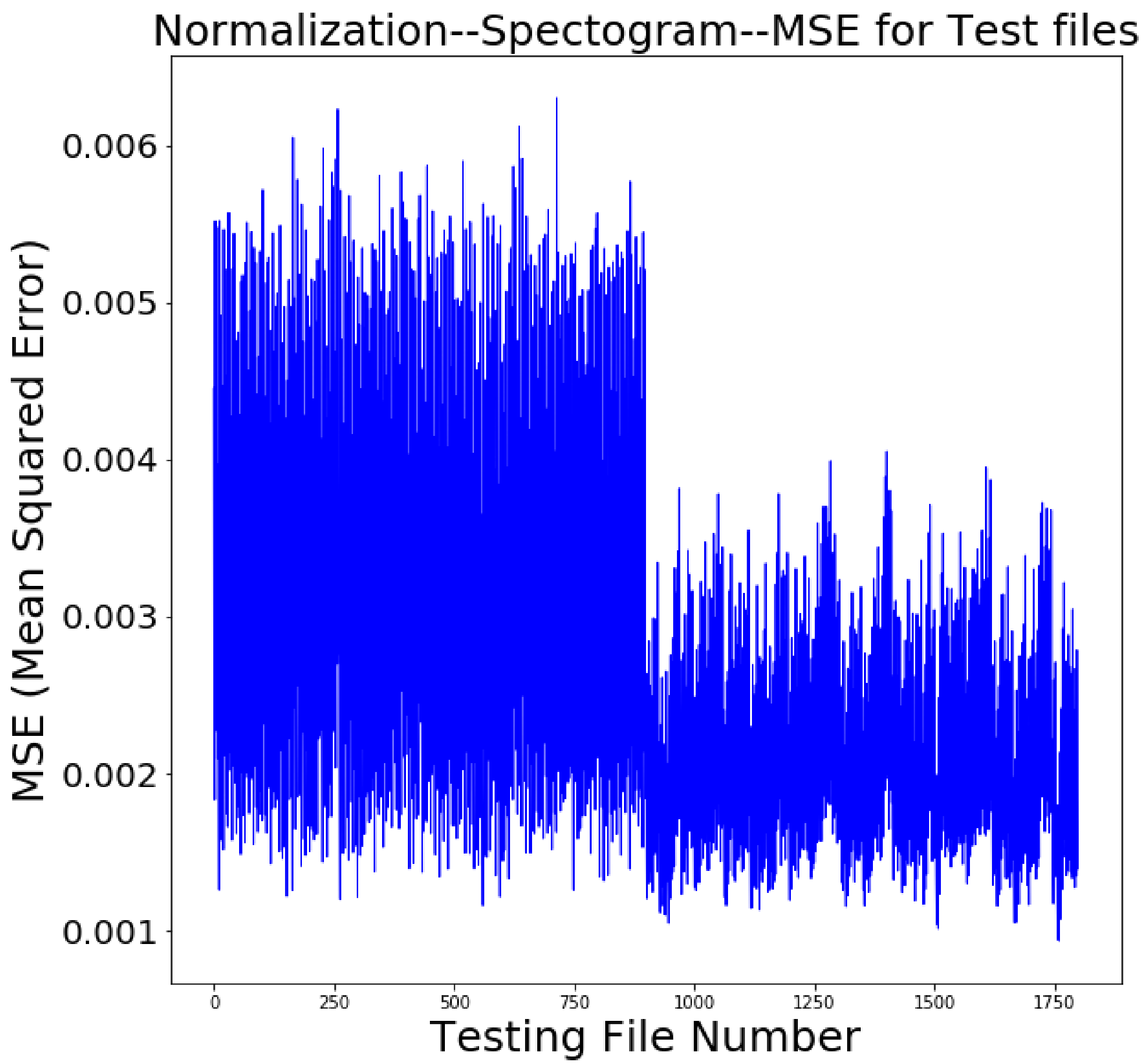
In [33]:
```
torch.save(best_model, './norm_spectogram.pth')
np.save('train_loss.npy',train_loss)
np.save('validation_loss.npy',validation_loss)
```

In [6]:
```
norm_spectogram = Net()
norm_spectogram.load_state_dict(torch.load('./norm_spectogram.pth'))
```

Out[6]:
```
<All keys matched successfully>
```

In [9]:
```
def generate_test_data(model,testpath,testfiles,outputpath,maximum,minimum,mean,s
    mse=[]
    for step in range(0,len(testfiles)):
        sx,sr = librosa.load(test_noisyPath +testfiles[step] ,sr=16000)
        S = librosa.stft(sx,n_fft=512,hop_length=160,win_length=320)
        abs_S = 10*np.log10(np.abs(S))
        phase=S/np.abs(S)
        if(input_type=="standardization"):
            abs_S=(abs_S-mean)/std
        elif(input_type=="normalization"):
            abs_S=(abs_S-minimum)/(maximum-minimum)
        else:
            print("ERROR")
        audio=torch.from_numpy(abs_S.astype('float32')).t()
        model.eval()
        mask=model(audio)
        mask=np.transpose(mask.cpu().data.numpy().squeeze())
        output=phase*(mask*abs_S)
        output=librosa.istft(output,hop_length=160,win_length=320)
        if(output.shape[0]<sx.shape[0]):
            output=np.pad(output, (0,sx.shape[0]-output.shape[0]),'constant')
        elif(output.shape[0]>sx.shape[0]):
            sx=np.pad(sx, (0,output.shape[0]-sx.shape[0]),'constant')
        mse.append(np.mean((output-sx)**2))
        output_filename=testfiles[step]
        librosa.output.write_wav(outputpath+output_filename,output,16000)
    return mse,len(testfiles)

import math
combined_file_arr=np.load('Test_noisy.npy')
combined_file_arr[combined_file_arr==-math.inf]=-100
combined_file_arr.shape
mean_X=np.mean(combined_file_arr,axis=1)
std_X=np.std(combined_file_arr,axis=1)
min_X=np.min(combined_file_arr,axis=1)
max_X=np.max(combined_file_arr,axis=1)

mse_1,length_testfiles=generate_test_data(norm_spectogram,test_noisyPath,test_no
```

In [15]:
```
mse_1,length_testfiles=generate_test_data(norm_spectogram,test_noisyPath,test_no
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_1,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('Normalization--Spectogram--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```



In [7]:

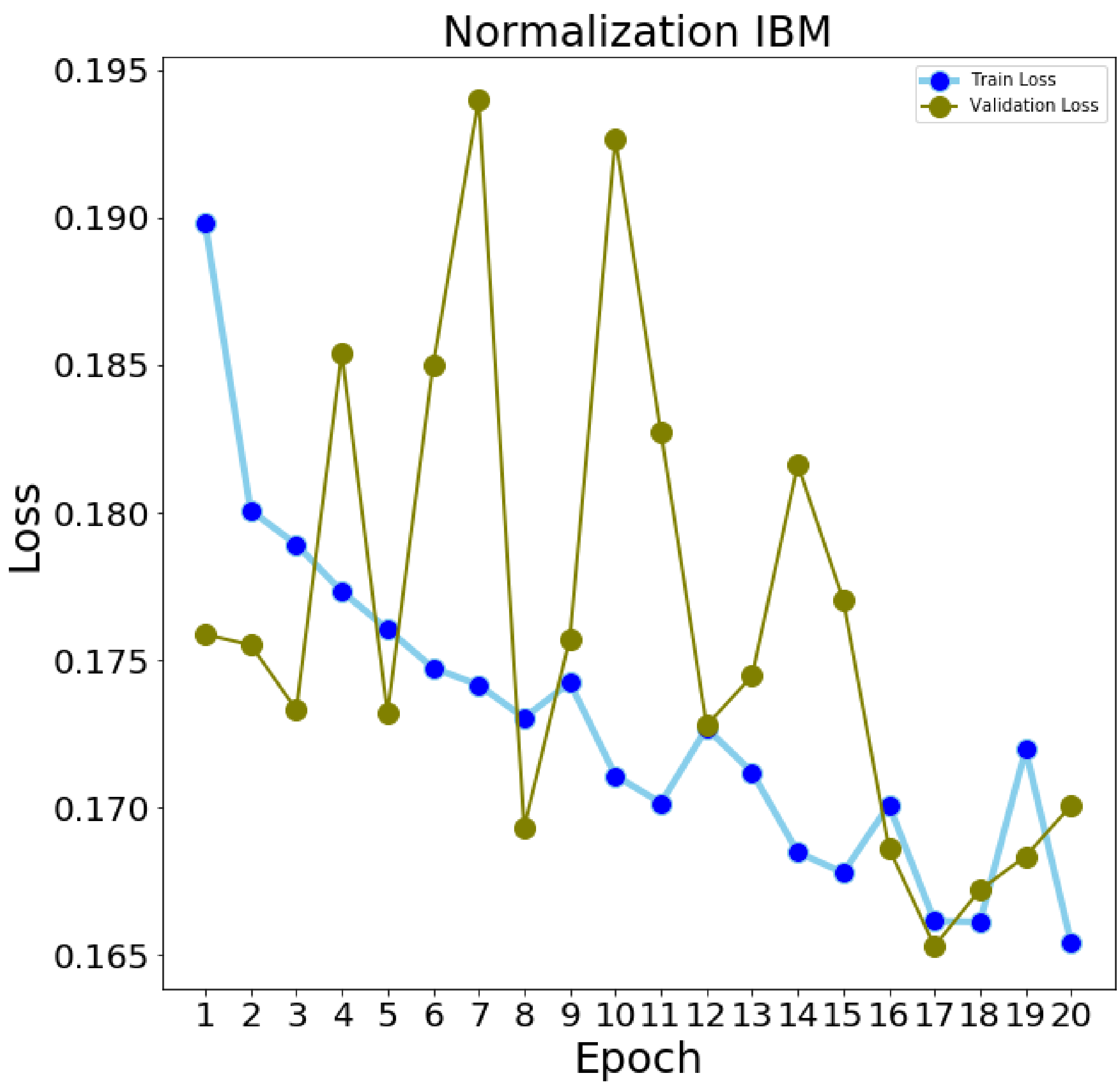
```
# NORMIZTION--> IBM

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='train_label_ibm.npy'
data_file='Normalization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4
label_file='dev_label_ibm.npy'
data_file='Normalization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000
train_loss_2,validation_loss_2,best_model_2=train_model(model,trainData,valData,
```

```
Epoch:14, Train Loss: 0.16779
Epoch:14, Valid Loss: 0.17704
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.17009
Epoch:15, Valid Loss: 0.16862
Epoch 16/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:16, Train Loss: 0.16615
Epoch:16, Valid Loss: 0.16530
Epoch 17/19
Train step:0/50
```

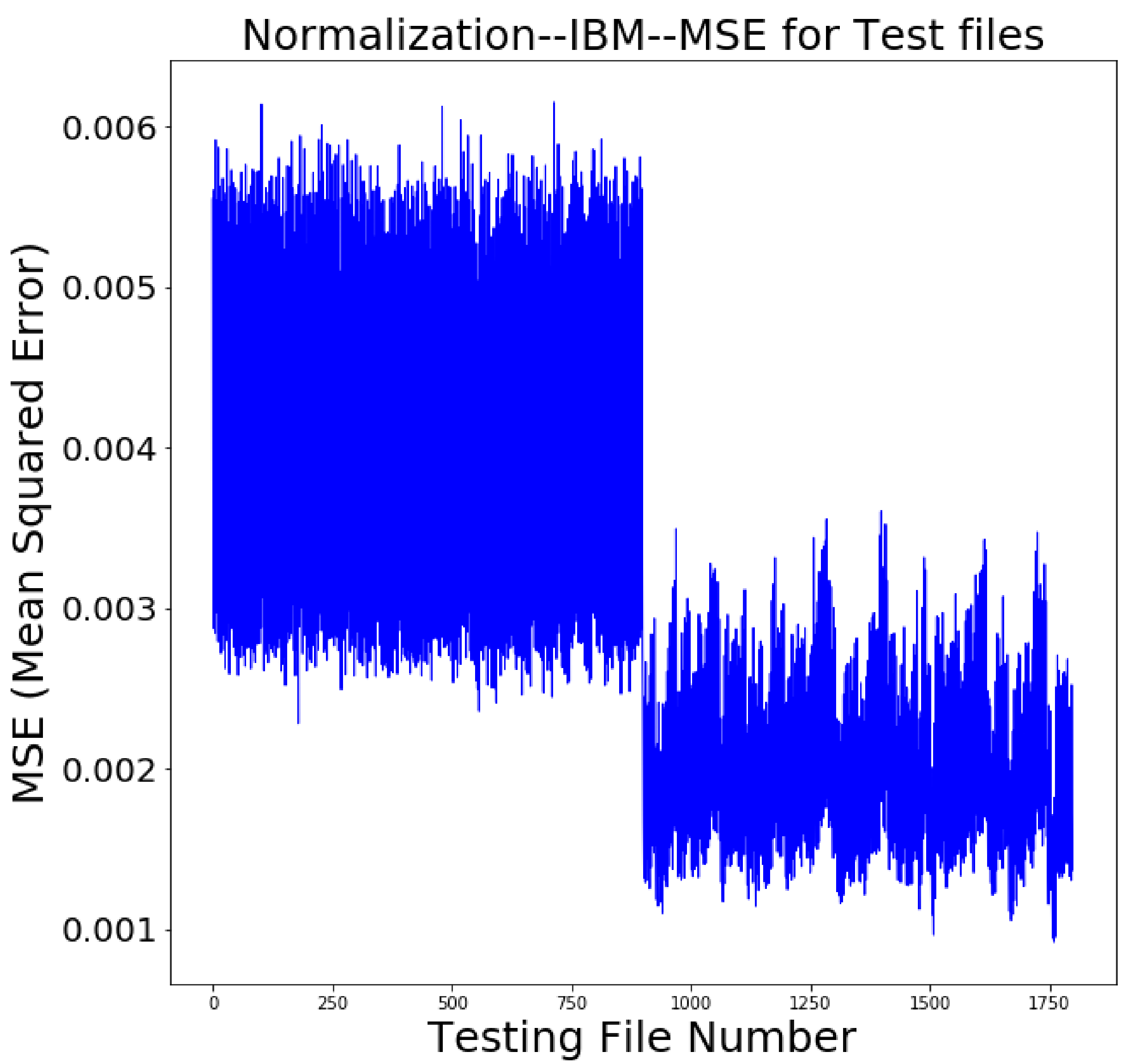
In [8]:

```
torch.save(best_model_2, './norm_ibm.pth')
np.save('train_loss_2.npy',train_loss_2)
np.save('validation_loss_2.npy',validation_loss_2)
```

In [82]:
```
train_loss_2=np.load('train_loss_2.npy')
validation_loss_2=np.load('validation_loss_2.npy')

import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss_2, marker='o', markerfacecolor='blue', markersiz
plt.plot(range(1,21),validation_loss_2, marker='o', color='olive',markersize=12,
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Normalization IBM",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
plt.show()
```



In [17]:
```
norm_ibm = Net()
norm_ibm.load_state_dict(torch.load('./norm_ibm.pth'))

mse_2,length_testfiles=generate_test_data(norm_ibm,test_noisyPath,test_noisySpee
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_2,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('Normalization--IBM--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```

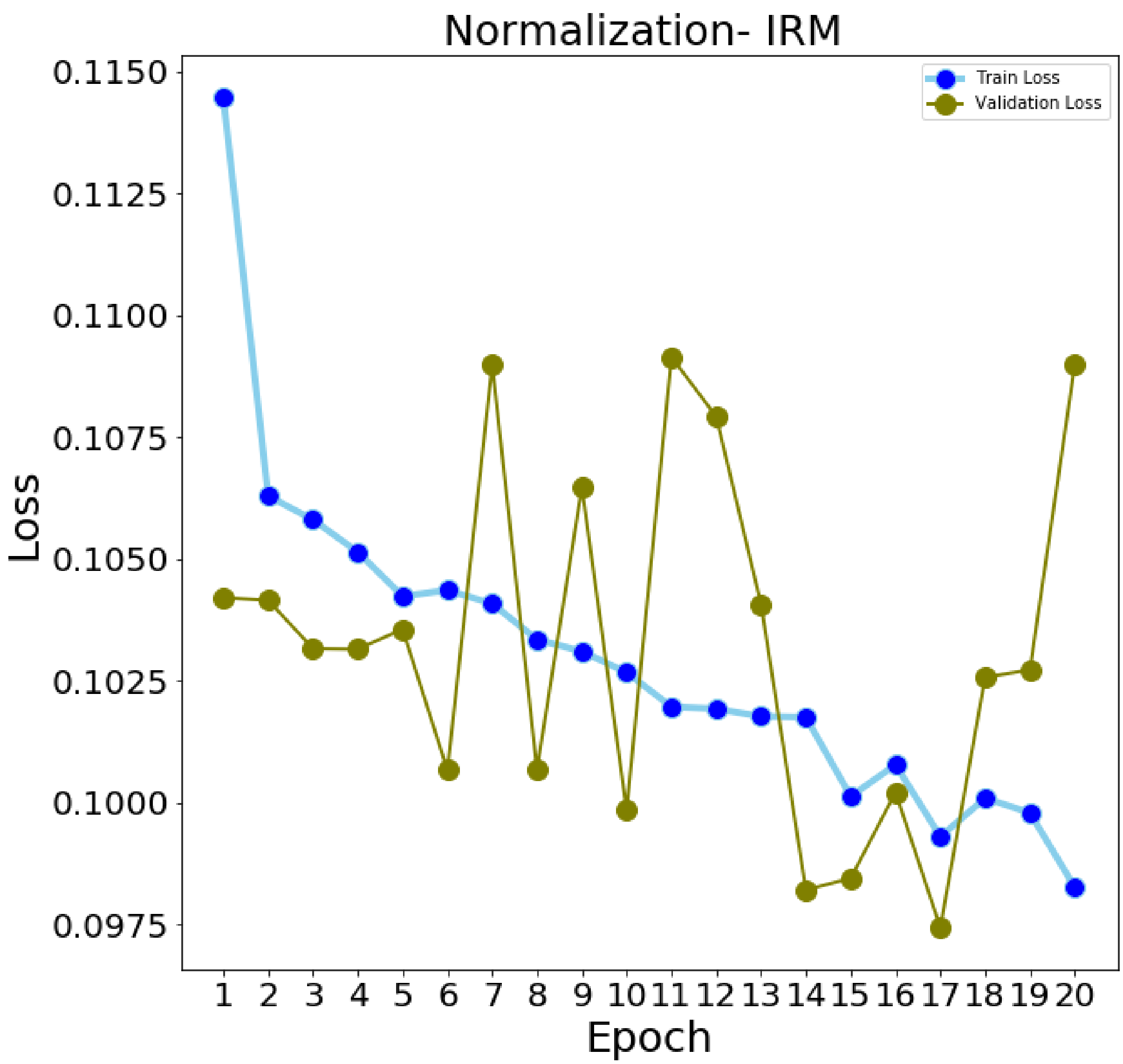


In [11]:
```
# NORMALIZATION--> IRM

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='train_label_irm.npy'
data_file='Normalization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4(
label_file='dev_label_irm.npy'
data_file='Normalization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000,
train_loss_3,validation_loss_3,best_model_3=train_model(model,trainData,valData,:

torch.save(best_model_3, './norm_irm.pth')
np.save('train_loss_3.npy',train_loss_3)
np.save('validation_loss_3.npy',validation_loss_3)
```

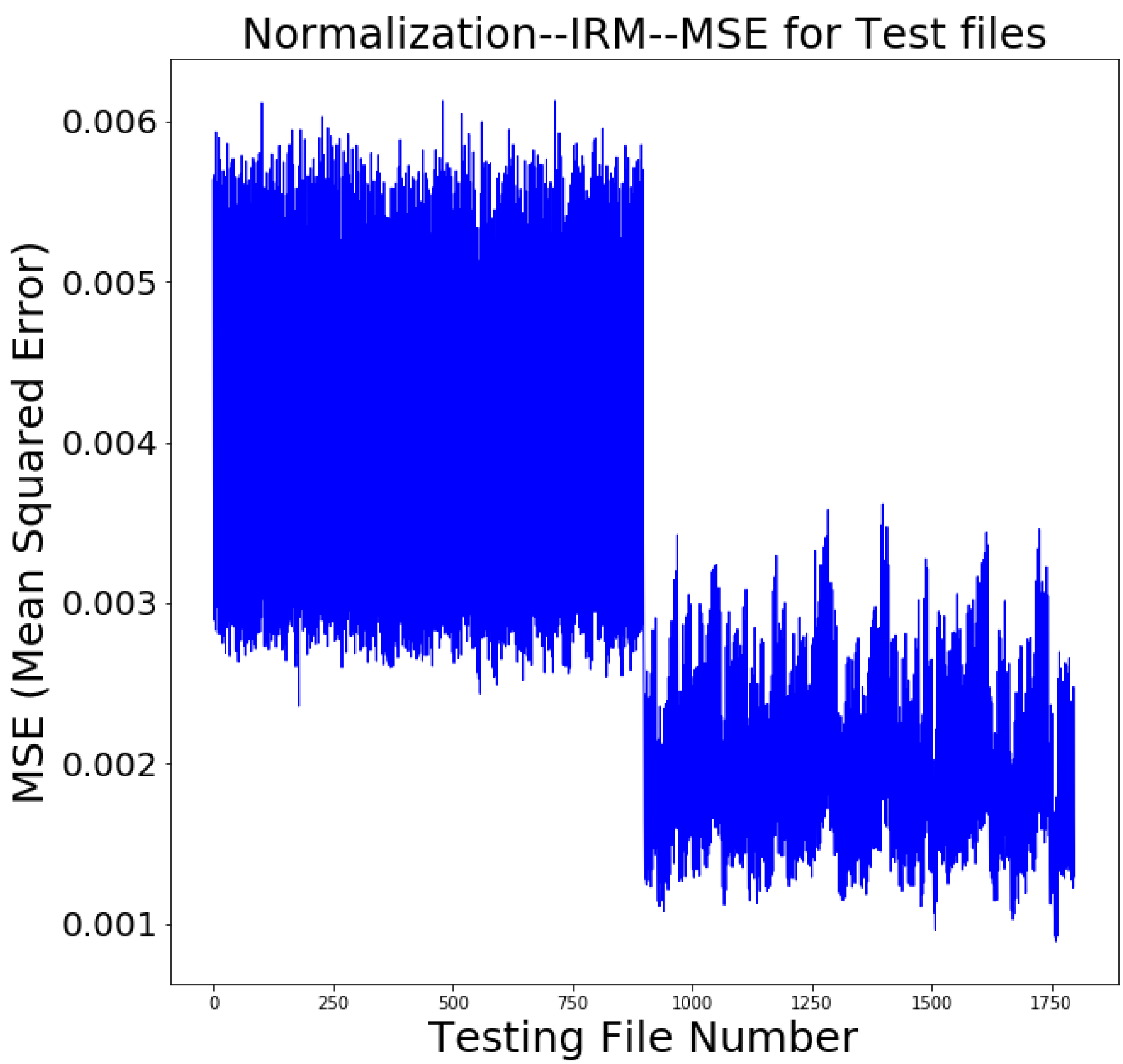
```
Epoch 14/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:14, Train Loss: 0.10013
Epoch:14, Valid Loss: 0.09844
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.10078
Epoch:15, Valid Loss: 0.10021
Epoch 16/19
Train step:0/50
Train step:15/50
```

In [84]:
```
train_loss_3=np.load('train_loss_3.npy')
validation_loss_3=np.load('validation_loss_3.npy')

import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss_3, marker='o', markerfacecolor='blue', markersiz
plt.plot(range(1,21),validation_loss_3, marker='o', color='olive',markersize=12,
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Normalization- IRM ",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
```

Out[84]:
```
Text(0, 0.5, 'Loss')
```



In [18]:
```
norm_irm = Net()
norm_irm.load_state_dict(torch.load('./norm_irm.pth'))

mse_3,length_testfiles=generate_test_data(norm_irm,test_noisyPath,test_noisySpee
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_3,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('Normalization--IRM--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```



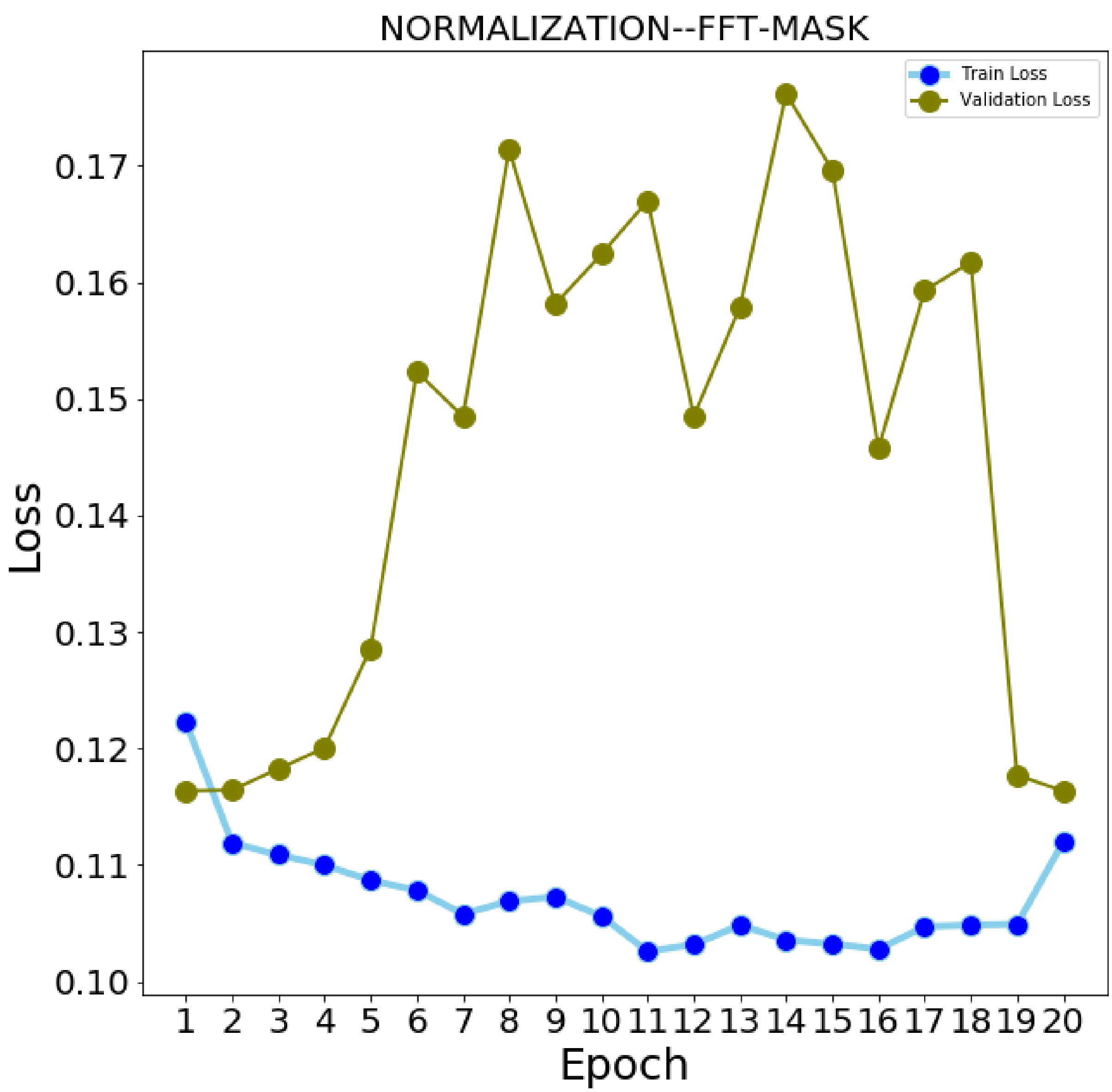
In [ ]:

In [19]:
```
# NORMALIZATION--> FFT_MASK

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='train_label_fft.npy'
data_file='Normalization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4
label_file='dev_label_fft.npy'
data_file='Normalization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000
train_loss_4,validation_loss_4,best_model_4=train_model(model,trainData,valData,

torch.save(best_model_4, './norm_fft.pth')
np.save('train_loss_4.npy',train_loss_4)
np.save('validation_loss_4.npy',validation_loss_4)
```

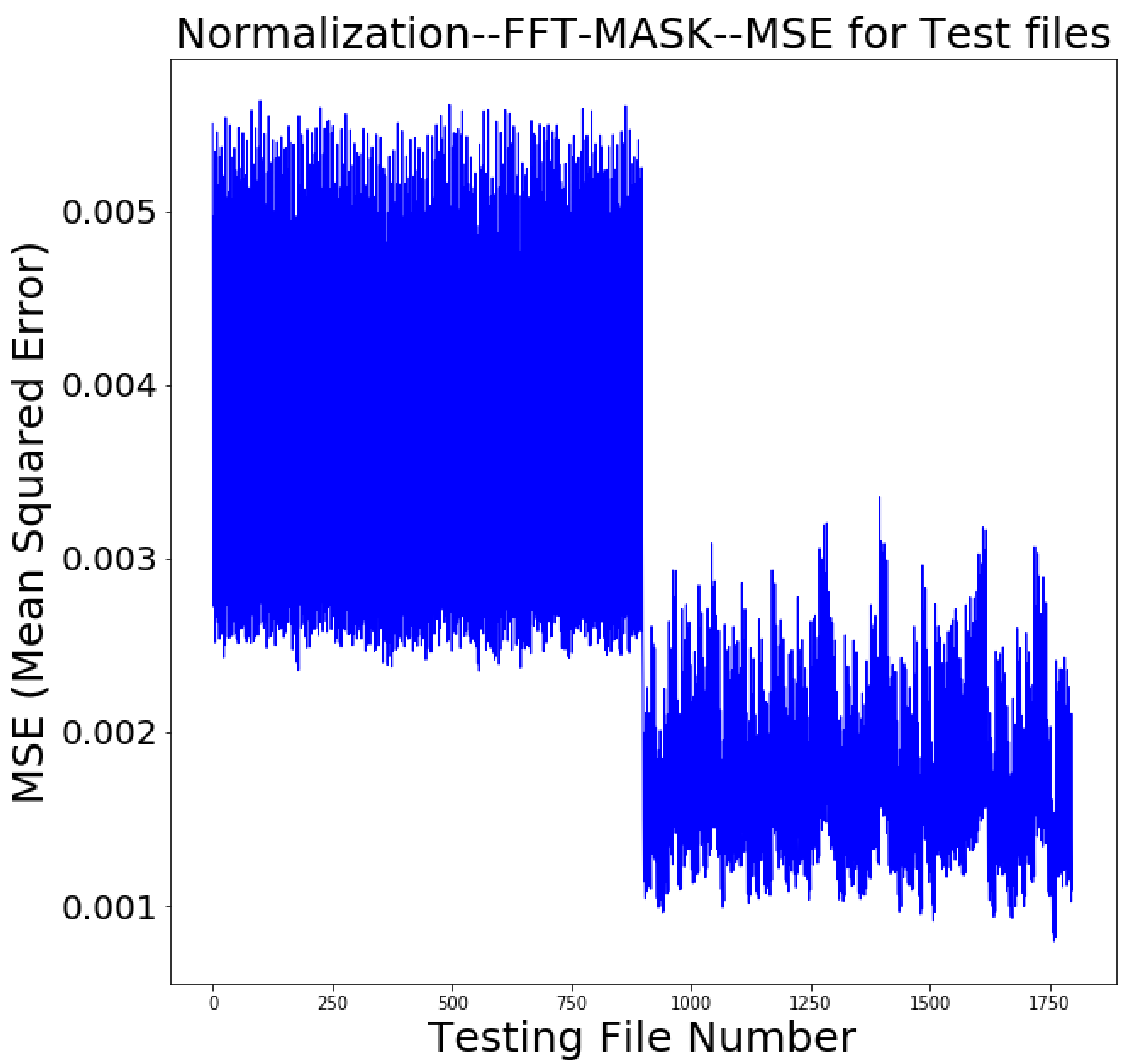
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: UserWarn
ing: nn.init.xavier_normal is now deprecated in favor of nn.init.xavier_norma
l_.
  app.launch_new_instance()
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:17: UserWarn
ing: nn.init.constant is now deprecated in favor of nn.init.constant_.
```

In [21]:
```
_loss_4=np.load('train_loss_4.npy')
ation_loss_4=np.load('validation_loss_4.npy')

t matplotlib.pyplot as plt
igure(figsize=(10,10))
lot(range(1,21),train_loss_4, marker='o', markerfacecolor='blue', markersize=12,
lot(range(1,21),validation_loss_4, marker='o', color='olive',markersize=12, linew
ticks(range(1,21),fontsize=20)
ticks(fontsize=20)
egend()
itle("NORMALIZATION--FFT-MASK",fontsize=20)
label("Epoch",fontsize=25)
label("Loss",fontsize=25)
```

Out[21]: Text(0, 0.5, 'Loss')



In [22]:
```
norm_fft = Net()
norm_fft.load_state_dict(torch.load('./norm_fft.pth'))

mse_4,length_testfiles=generate_test_data(norm_fft,test_noisyPath,test_noisySpee
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_4,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('Normalization--FFT-MASK--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```

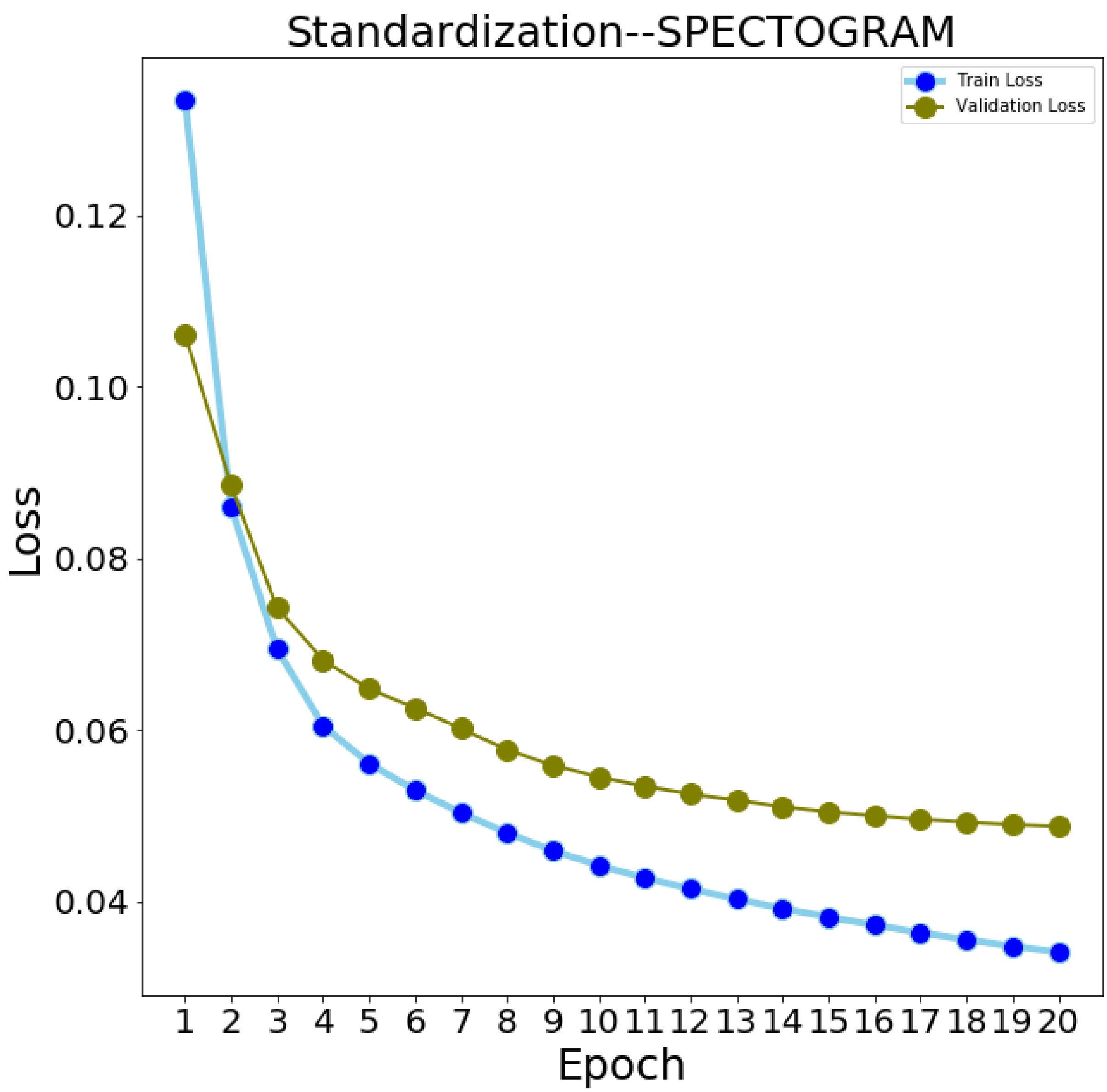


In [9]:

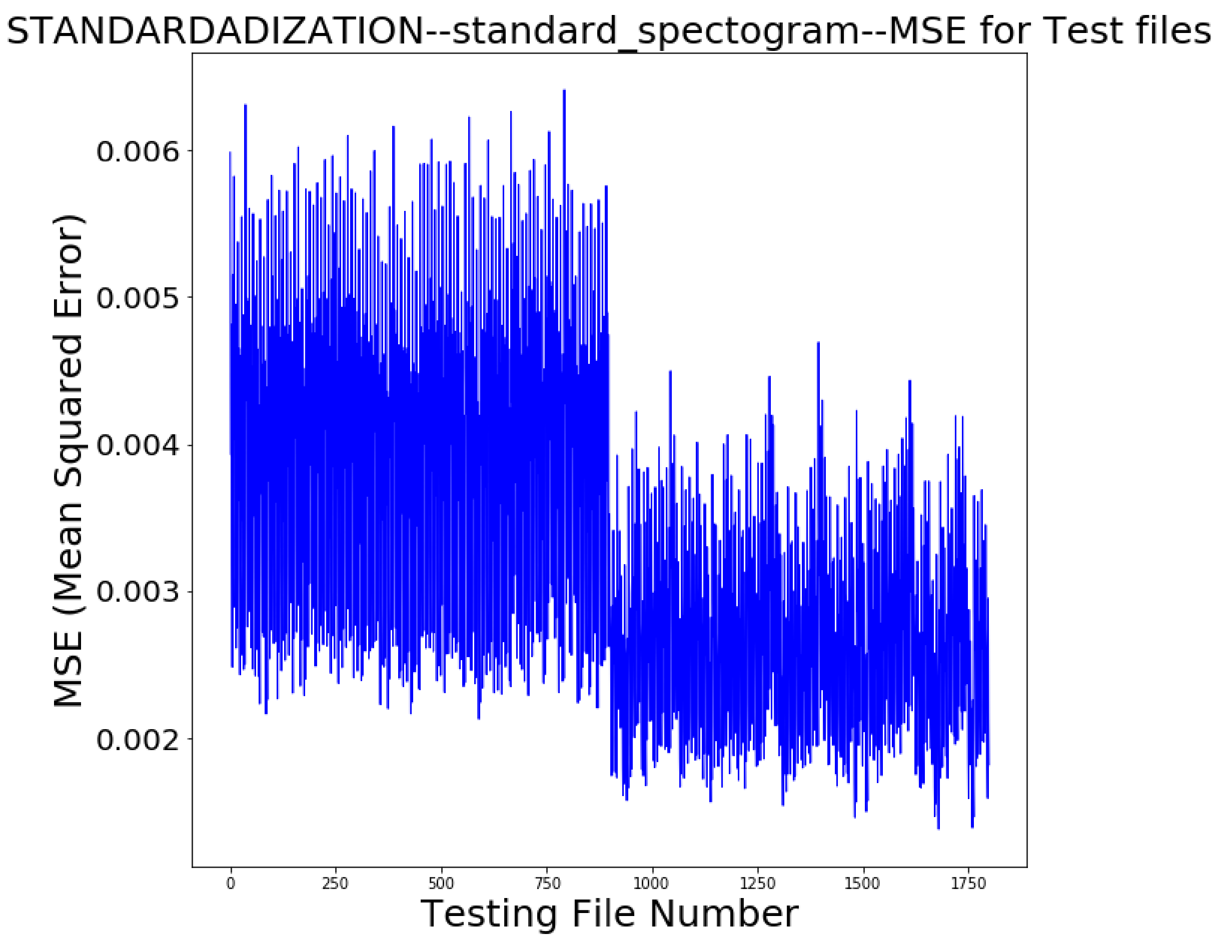
```
# Standardization SPECTOGRAM

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='Train_clean.npy'
data_file='Standardization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4(
label_file='Dev_clean.npy'
data_file='Standardization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000,
train_loss,validation_loss,best_model=train_model(model,trainData,valData,20)
train_loss_5,validation_loss_5,best_model_5=train_loss,validation_loss,best_model

torch.save(best_model_5, './standard_spectogram.pth')
np.save('train_loss_5.npy',train_loss_5)
np.save('validation_loss_5.npy',validation_loss_5)
```

```
Epoch:14, Train Loss: 0.03818
Epoch:14, Valid Loss: 0.05046
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.03727
Epoch:15, Valid Loss: 0.05004
Epoch 16/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:16, Train Loss: 0.03637
Epoch:16, Valid Loss: 0.04960
Epoch 17/19
Train step:0/50
```

In [14]:
```
train_loss_5=np.load('train_loss_5.npy')
validation_loss_5=np.load('validation_loss_5.npy')
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss_5, marker='o', markerfacecolor='blue', markersiz
plt.plot(range(1,21),validation_loss_5, marker='o', color='olive',markersize=12,
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Standardization--SPECTOGRAM",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
plt.show()
```



In [23]:
```
standard_spectogram = Net()
standard_spectogram.load_state_dict(torch.load('./standard_spectogram.pth'))

mse_5,length_testfiles=generate_test_data(standard_spectogram,test_noisyPath,tes
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_5,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('STANDARDADIZATION--standard_spectogram--MSE for Test files',fontsize=
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```

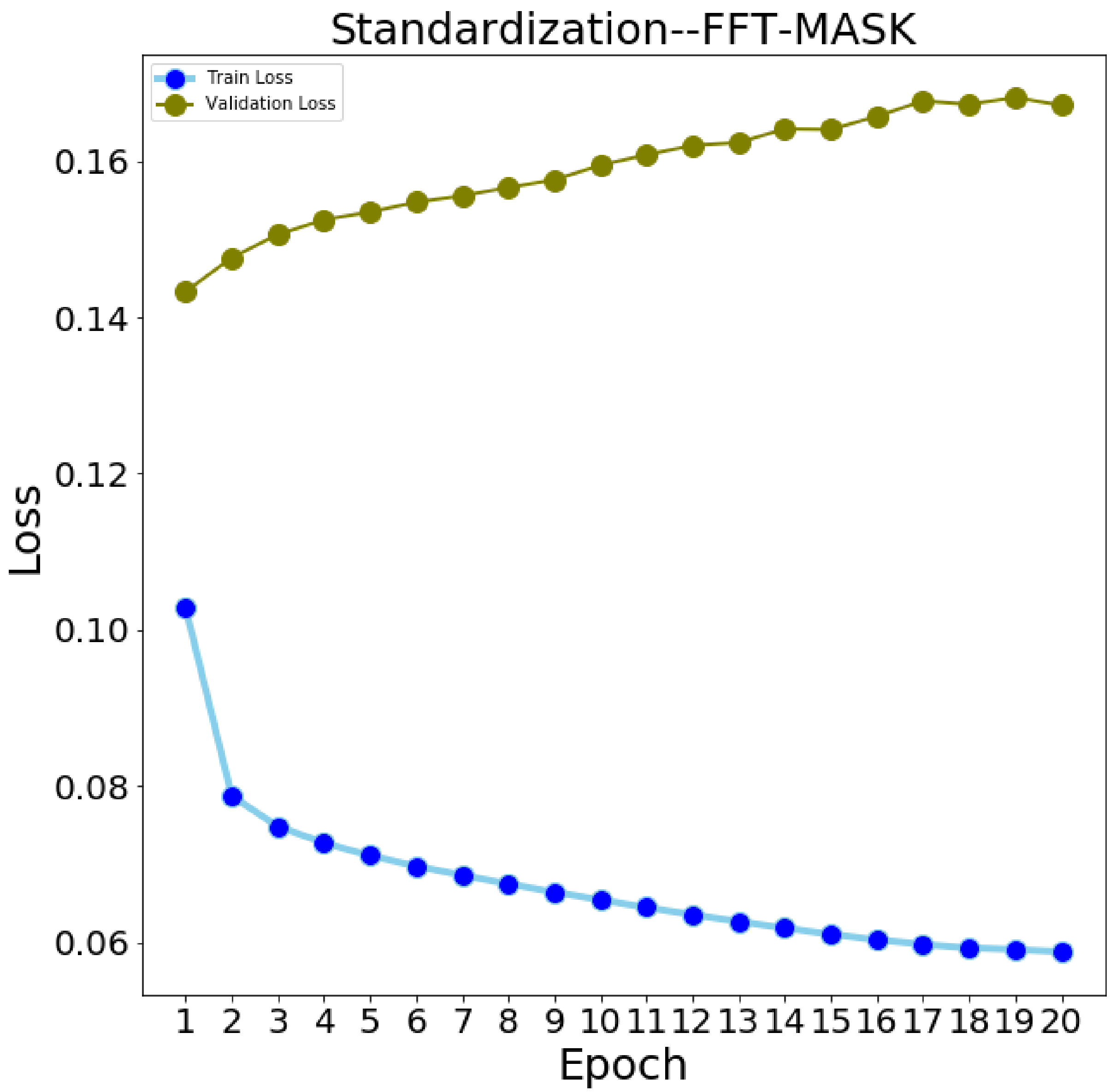


In [6]:
```
# Standardization--> FFT_MASK

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='train_label_fft.npy'
data_file='Standardization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4(
label_file='dev_label_fft.npy'
data_file='Standardization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000,
train_loss_6,validation_loss_6,best_model_6=train_model(model,trainData,valData,

torch.save(best_model_6, './standard_fft.pth')
np.save('train_loss_6.npy',train_loss_6)
np.save('validation_loss_6.npy',validation_loss_6)
```

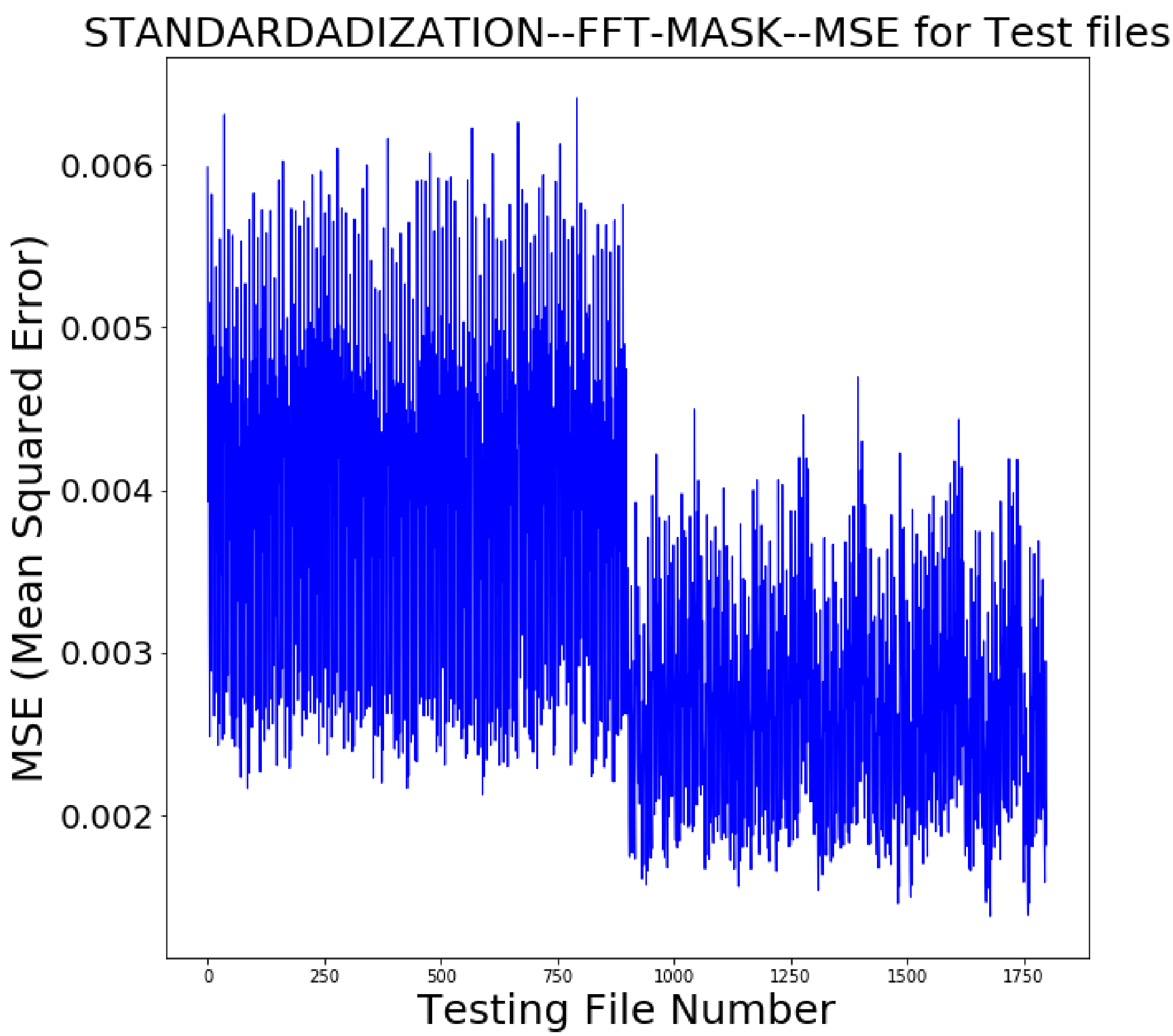
```
Epoch:14, Train Loss: 0.06106
Epoch:14, Valid Loss: 0.16404
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.06034
Epoch:15, Valid Loss: 0.16573
Epoch 16/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:16, Train Loss: 0.05976
Epoch:16, Valid Loss: 0.16769
Epoch 17/19
Train step:0/50
```

In [9]:
```
train_loss_6=np.load('train_loss_6.npy')
validation_loss_6=np.load('validation_loss_6.npy')
```

In [10]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss_6, marker='o', markerfacecolor='blue', markersiz
plt.plot(range(1,21),validation_loss_6, marker='o', color='olive',markersize=12,
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Standardization--FFT-MASK",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
plt.show()
```



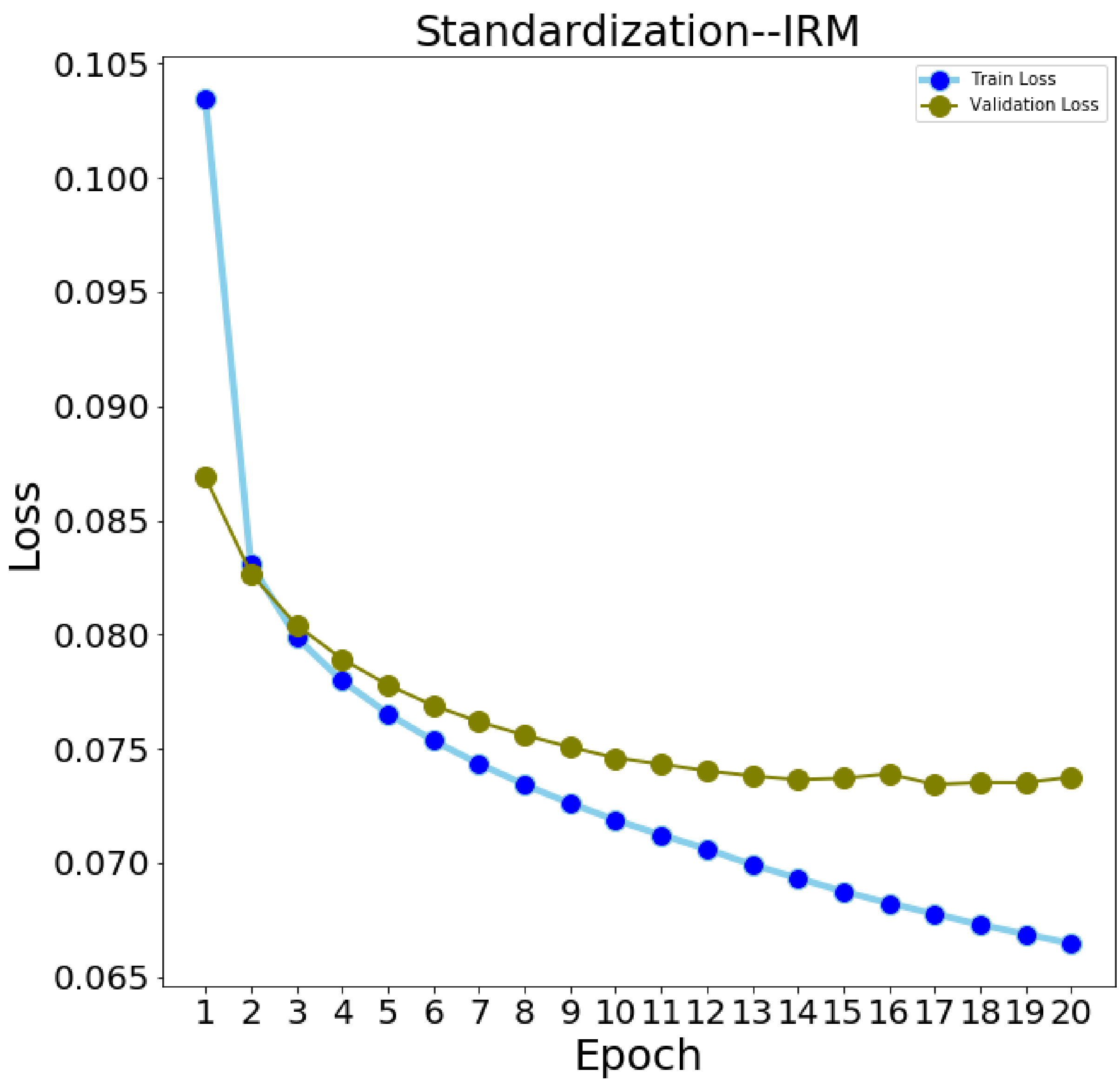
In [36]:
```
standard_fft = Net()
standard_fft.load_state_dict(torch.load('./standard_fft.pth'))

mse_6,length_testfiles=generate_test_data(standard_fft,test_noisyPath,test_noisy
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_6,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('STANDARDADIZATION--FFT-MASK--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```



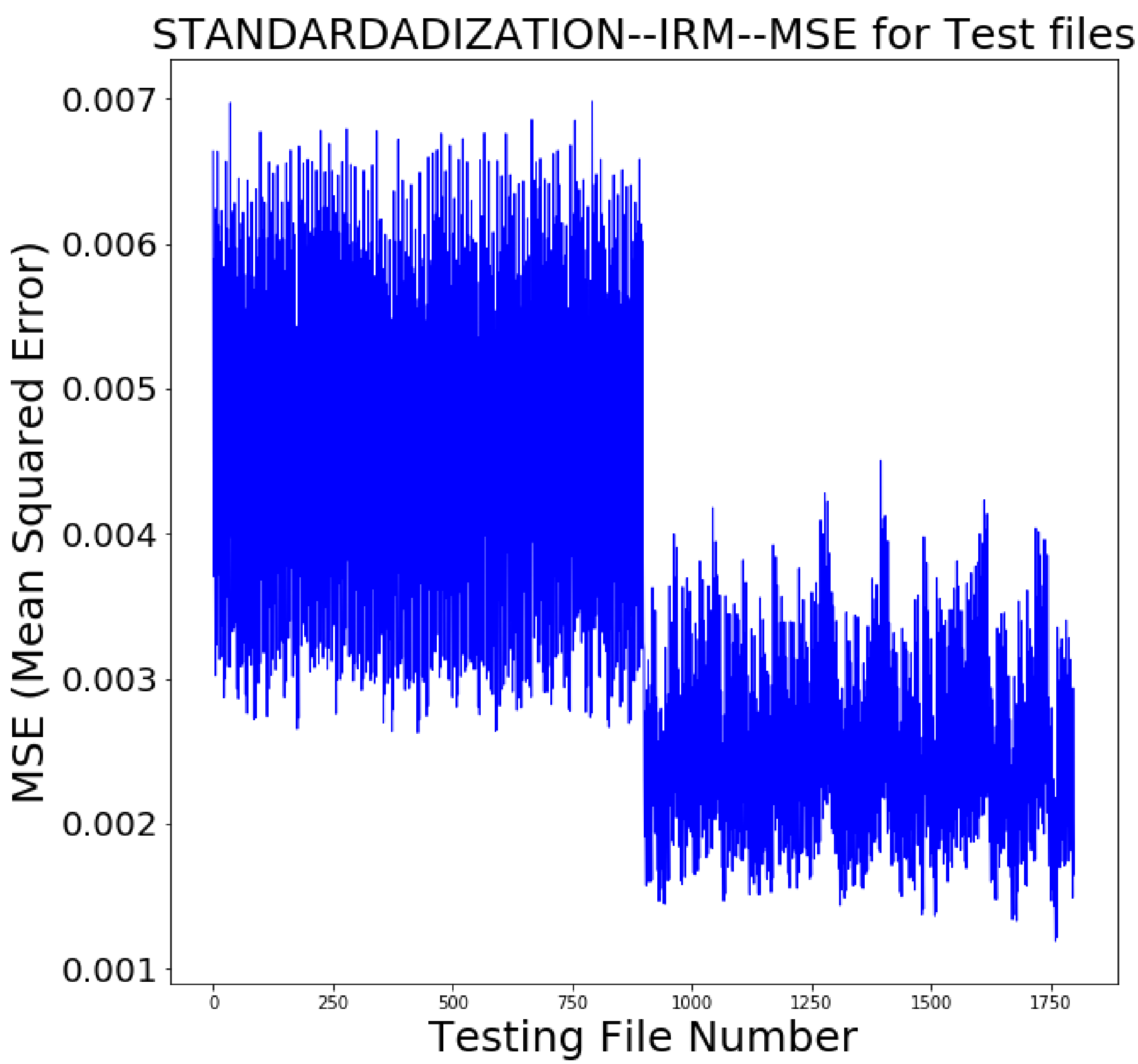
In [6]:
```
# Standardization--> IRM

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='train_label_irm.npy'
data_file='Standardization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4(
label_file='dev_label_irm.npy'
data_file='Standardization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000,
train_loss_7,validation_loss_7,best_model_7=train_model(model,trainData,valData,:

torch.save(best_model_7, './standard_irm.pth')
np.save('train_loss_7.npy',train_loss_7)
np.save('validation_loss_7.npy',validation_loss_7)
```

```
Valid step:0/5
Epoch:14, Train Loss: 0.06874
Epoch:14, Valid Loss: 0.07370
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.06823
Epoch:15, Valid Loss: 0.07389
Epoch 16/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:16, Train Loss: 0.06776
Epoch:16, Valid Loss: 0.07343
Epoch 17/19
Train step:0/50
```

In [13]:
```
train_loss_7=np.load('train_loss_7.npy')
validation_loss_7=np.load('validation_loss_7.npy')

import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss_7, marker='o', markerfacecolor='blue', markersiz
plt.plot(range(1,21),validation_loss_7, marker='o', color='olive',markersize=12,
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Standardization--IRM",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
plt.show()
```



In [35]:
```
standard_irm = Net()
standard_irm.load_state_dict(torch.load('./standard_irm.pth'))

mse_7,length_testfiles=generate_test_data(standard_irm,test_noisyPath,test_noisyS
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_7,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('STANDARDADIZATION--IRM--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```

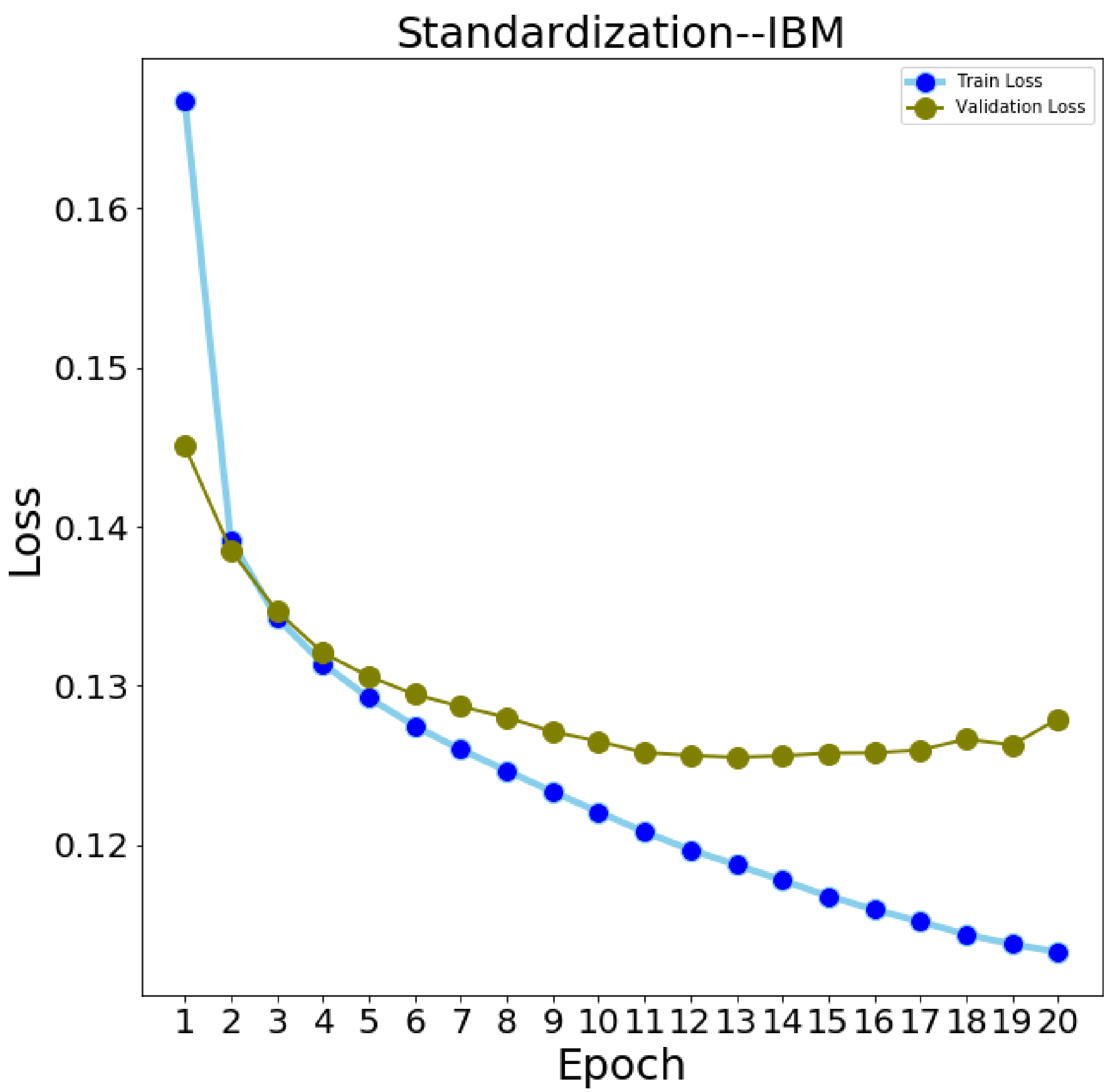


In [6]:
```
# Standardization--> IBM

model = Net()
model.apply(weights)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters())

#device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model.cuda()
#model = #
#criterion.cuda()

label_file='train_label_ibm.npy'
data_file='Standardization_Train_noisy.npy'
trainData = data.DataLoader(trainDataLoader(label_file,data_file),batch_size = 4(
label_file='dev_label_ibm.npy'
data_file='Standardization_Dev_noisy.npy'
valData = data.DataLoader(valDataLoader(label_file,data_file),batch_size = 40000,
train_loss_8,validation_loss_8,best_model_8=train_model(model,trainData,valData,
```

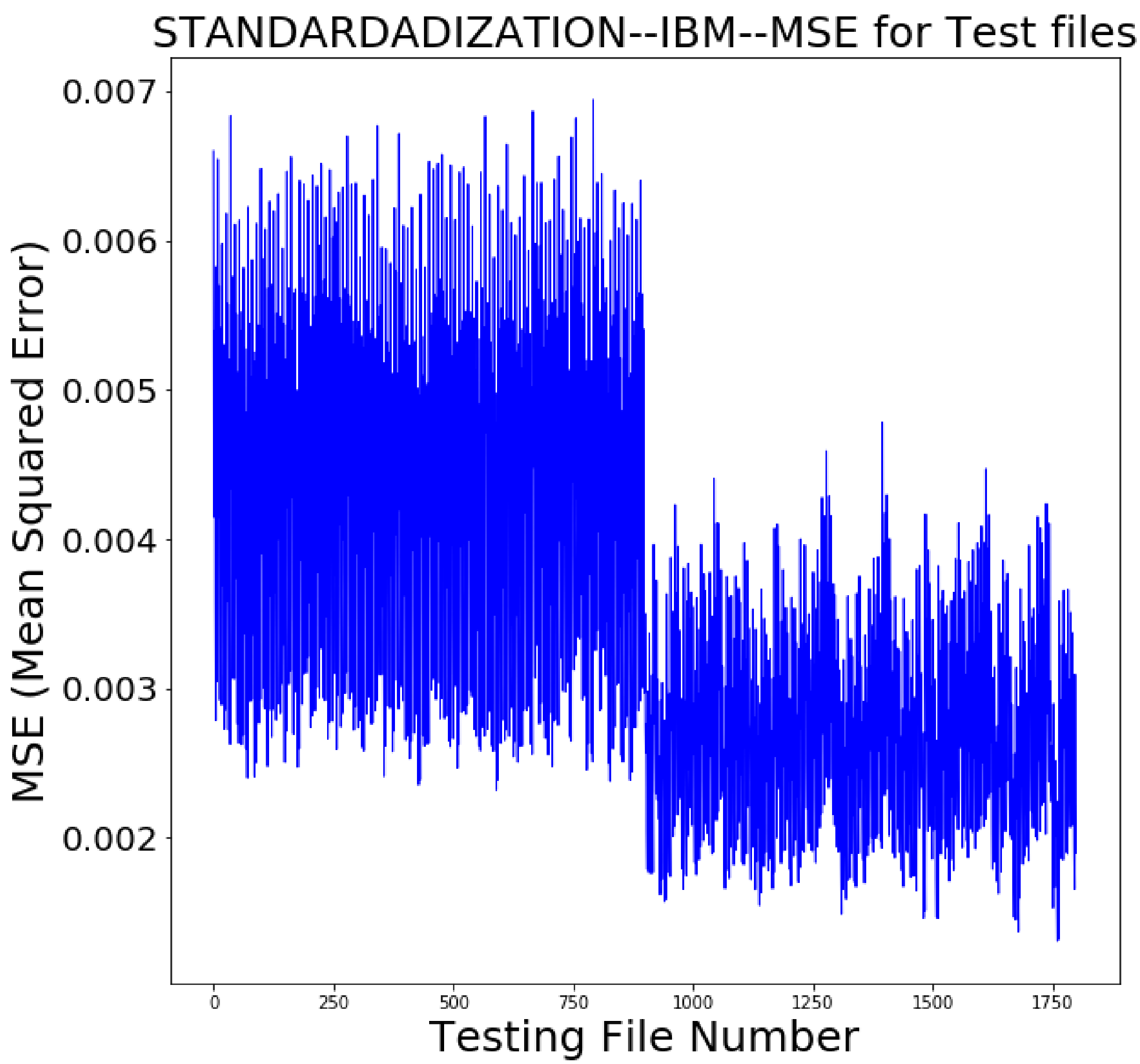
```
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:14, Train Loss: 0.11676
Epoch:14, Valid Loss: 0.12576
Epoch 15/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
Valid step:0/5
Epoch:15, Train Loss: 0.11592
Epoch:15, Valid Loss: 0.12577
Epoch 16/19
Train step:0/50
Train step:15/50
Train step:30/50
Train step:45/50
```

In [ ]:
```
torch.save(best_model_8, './standard_ibm.pth')
np.save('train_loss_8.npy',train_loss_8)
np.save('validation_loss_8.npy',validation_loss_8)
```

In [11]:
```
train_loss_8=np.load('train_loss_8.npy')
validation_loss_8=np.load('validation_loss_8.npy')

import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(1,21),train_loss_8, marker='o', markerfacecolor='blue', markersiz
plt.plot(range(1,21),validation_loss_8, marker='o', color='olive',markersize=12,
plt.xticks(range(1,21),fontsize=20)
plt.yticks(fontsize=20)
plt.legend()
plt.title("Standardization--IBM",fontsize=25)
plt.xlabel("Epoch",fontsize=25)
plt.ylabel("Loss",fontsize=25)
plt.show()
```



In [31]:
```
standard_ibm = Net()
standard_ibm.load_state_dict(torch.load('./standard_ibm.pth'))

mse_8,length_testfiles=generate_test_data(standard_ibm,test_noisyPath,test_noisy
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.plot(range(length_testfiles),mse_8,color='blue', linewidth=1)
#plt.xticks(range(length_testfiles),fontsize=20)
plt.yticks(fontsize=20)
plt.title('STANDARDADIZATION--IBM--MSE for Test files',fontsize=25)
plt.xlabel("Testing File Number",fontsize=25)
plt.ylabel("MSE (Mean Squared Error)",fontsize=25)
plt.show()
```



Question 2 (continued) Discuss how the different DNNs perform. Also, discuss how data normalization vs data standardization impacts performance.

```
I tried creating multiple (nearly 4.5 Million) .npy files but it did not turn
out well for training such a long number of files to create another files. Thus
I came up with my own approach where we combine whole 4.5 Million training data
into ONE 2 dimensional array. This approach was much more efficient for
training and storing/reading files/creating intermediate files from present
files.

How different Neural Networks perform?

IRM was my favourite one. FFT Mask was comparable to it.
Standard Spectogram and Standard IRM, then Normalized IRM and then Normalized
FFT-Mask is the performance wise result of Neural Networks. Overall I would
suggest that IRM was better in most of the cases.

Data Normalization vs Data Standardization:
For me, clearly Data Standardization performed better in below aspects:

1. The IBM, IRM and Spectogram of Standardized Data is very very good as it has
training and validation loss consistently decreasing for all 20 epochs and both
are decreasing side by side without any ups and downs. (An exception of Data
Standardization occurs when we have FFT-Mask. In this case validation loss is
not decreasing and when it is not decreasing and we try to fit more and more
training data, we can see that overfitting occurs.)

2. A key  thing to notice about Data Standardization is " All loss curves for
all training and all validation curves are SMOOTH in nature. ie they have
smooth curvature and not ups and downs. " Data Standardization thus is very
good as it creates domain in such a distribution that the gradients are mostly
in correct direction. (Moving towards center of minimum directly instead of
roaming along sides and indirectly reaching towards minima point.)

3. Normalization also helps us in scaling the values to similar dimension
instead of too much sparse values but it is not effective as much as
Standardization approach. This is because we can still have skewness in data
after performing Normalilzation as it depends on minimum and maximum values.
For Ex: In House prices, if one house is of 3Billion Dollars, it creates lot of
skewness when data is normalized using min and max and this leads to almost all
house prices near 0 and One 3 B $ house as 1. For our dataset, Normalization
did not perform well enough is my conclusion.

4. Training loss and validation loss for Data Standardization is less as
compared to Data Normalization.
```

Question 3 (continued): Also, listen to the signals and explain how the performance varies audibly. Also submit two sound examples (clean speech, noisy speech,

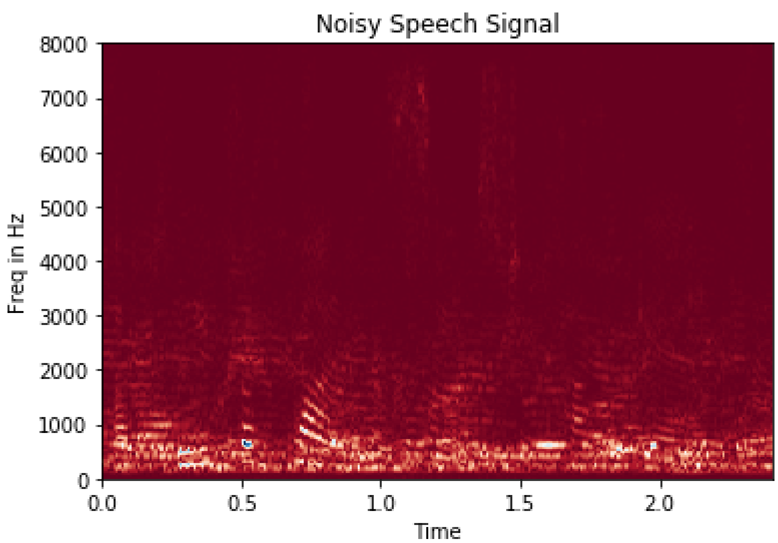
enhanced speech) for each training target. Describe how the different targets perform at speech enhancement, both computationally and perceptually.

```
For all the Mean Squared Error plots, i could clearly see that MSE was very
very less for all the test speech.

A very good trend that is observed is that the difference between male and
female speeches. All male samples had same pattern and all female samples had
same pattern among them. Also one group had higher MSE than other.

Computationally, IBM should be cheaper than other models but unfortunately I
could not see any major changes in computations practially. Almost all models
performed in same time.

By listening to many of the outputs, IRM performed better than other. FFT-Mask
was also somewhat better in hearing. Perceptually, IRM and FFT Mask performed
almost similar. Moreover, all models had almost similar MSE ranging from 0.001
to 0.006. Overall I would say that IRM and FFT mask performed almost similar
and hearing quality of signal was also good.
```

Q1. (continued) Take one of your noisy speech signals (and corresponding speech and noise components) and generate the IBM and IRM masks. Plot the spectrograms for the noisy speech signal, clean speech component, noise component and appropriately label all axis. Also, generate plots for the corresponding IBM and IRM.

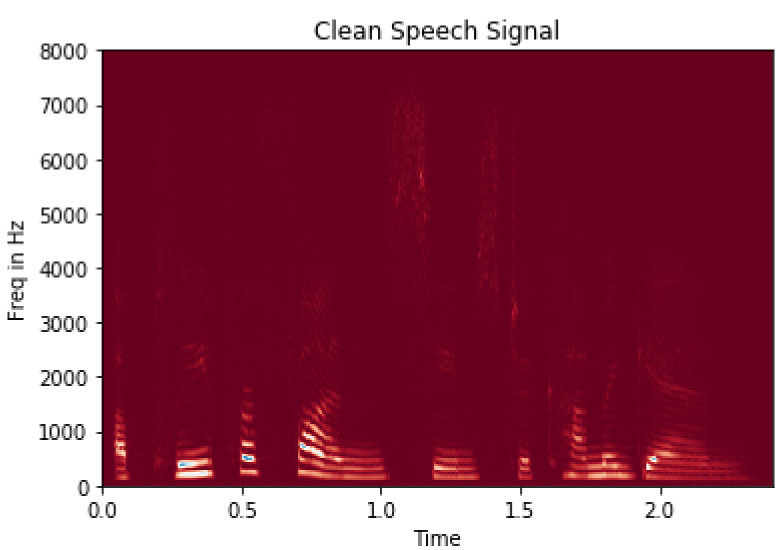
In [40]:
```
noisy_signal_filepath='PREPARED_DATASET/TEST/l61s01__0_0.wav'
sx,sr = librosa.load(noisy_signal_filepath,sr=None)
noisy_signal = np.abs(librosa.stft(sx,n_fft=512,hop_length=160,win_length=320))
extent=[0,len(sx)/sr,0,8000]
plt.title("Noisy Speech Signal")
plt.ylabel("Freq in Hz")
plt.xlabel("Time")
plt.imshow(noisy_signal,origin='lowest',aspect='auto',extent=extent,cmap='RdBu')
```

Out[40]: <matplotlib.image.AxesImage at 0x1cc95350780>



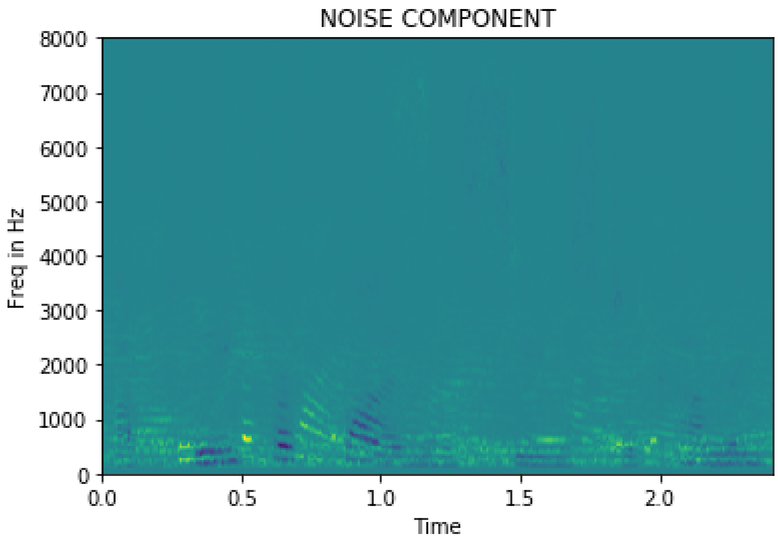
In [74]:
```
clean_filepath='PREPARED_DATASET/CLEAN/TEST/l61s01.wav'
sx,sr = librosa.load(clean_filepath,sr=None)
clean = np.abs(librosa.stft(sx,n_fft=512,hop_length=160,win_length=320))
extent=[0,len(sx)/sr,0,8000]
plt.title("Clean Speech Signal")
plt.ylabel("Freq in Hz")
plt.xlabel("Time")
plt.imshow(clean,origin='lowest',aspect='auto',extent=extent,cmap='RdBu')
```

Out[74]: <matplotlib.image.AxesImage at 0x1cc8d76def0>



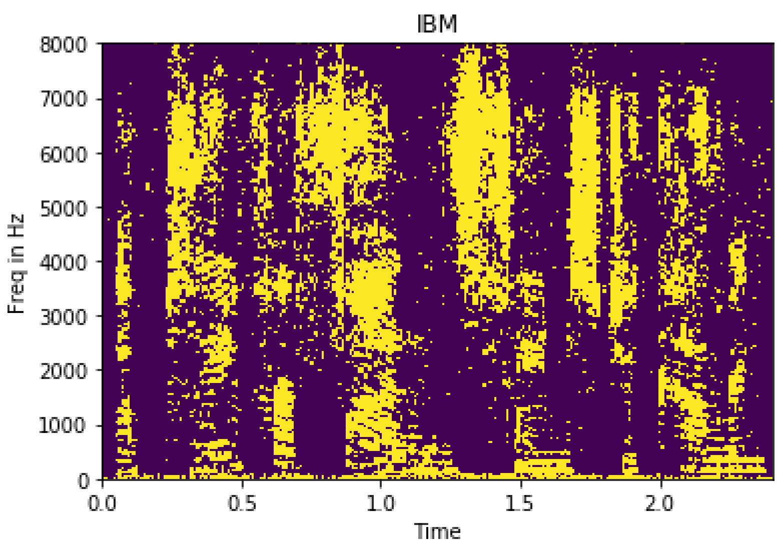
In [75]:
```
noise=noisy_signal-clean
extent=[0,len(sx)/sr,0,8000]
plt.title("NOISE COMPONENT")
plt.ylabel("Freq in Hz")
plt.xlabel("Time")
plt.imshow(noise,origin='lowest',aspect='auto',extent=extent)
```

Out[75]: <matplotlib.image.AxesImage at 0x1cc93d414e0>



In [78]:
```
def IBM(noisy_speech,clean_speech):
    noise=noisy_speech-clean_speech
    mask=clean_speech
    mask[clean_speech>=noise]=1
    mask[clean_speech<noise]=0
    return mask
def IRM(noisy_speech,clean_speech):
    noise=noisy_speech-clean_speech
    speech_energy=np.array(clean_speech)**2
    noise=np.array(noise)**2
    irm = np.sqrt(speech_energy / (noise + speech_energy))
    return irm

ibm=IBM(noisy_signal,clean)
extent=[0,len(sx)/sr,0,8000]
plt.title("IBM")
plt.ylabel("Freq in Hz")
plt.xlabel("Time")
plt.imshow(ibm,origin='lowest',aspect='auto',extent=extent)
```

Out[78]: `<matplotlib.image.AxesImage at 0x1cc93e335f8>`



In [79]:
```
irm=IRM(noisy_signal,clean)
extent=[0,len(sx)/sr,0,8000]
plt.title("IRM")
plt.ylabel("Freq in Hz")
plt.xlabel("Time")
plt.imshow(irm,origin='lowest',aspect='auto',extent=extent)
```

Out[79]: <matplotlib.image.AxesImage at 0x1cc93e831d0>