https://colab.research.google.com/drive/1iChV_DmAbHk_KZrwapAAlx1ERzMYkc7d?fbclid=IwAR0
8mPeepJyLnFZ7SkdlK-qY0#scrollTo=BYHVVrbSW9C3

Important notes

- The data used in this project is based on the Google Speech Commands Dataset. The Googl
the TensorFlow and AIY teams to showcase the speech recognition example using the Tens
one-second-long duration. Each clip contains one of the 30 different words spoken by thous
loading times only 10 of the 35 words has been included
- This notebook has incorporated the use of Google Drive to help working with big amount of
used in this project and experiment themselves, we highly recommend to set up a Google Dr
setting up a Google drive the following dataset is needed:

https://drive.google.com/drive/folders/1daJSE_-uv3foWKIJXjhGdZZQcQ1cIpOr

Click on the link

Click on the folder speech_command_recognition

Copy to own drive and do not put it in any subfolders

Afterwards simply run all the code below.

Gennerating spectorgrams

To gennerate the spectrograms we use the following colab code

https://colab.research.google.com/drive/1ysqM5inDoOSerYTOM-nLnK0P4ciZUuWg#scrollTo=Bd

Gennerating the spectrograms will be done differently in a custom dataset but for the datasets in t
dataset used below we use 10 words where the total dataset has 35 but takes alot longer to load.

A Tutorial of Speech Recognition

Artificial intelligence is becoming more and more a part of our daily lives. There is a lot of confusi
recognition. They refer to the same thing but are two different things. In this Tutorial, speech recog

Speech recognition is when a computer or machine can safely identify the words spoken by anyon
machine is able to recognize and identify a particular voice. For example, Siri or Amazon's Alexa. S
years, but through Deep Learning it has many innovative uses in our daily lives

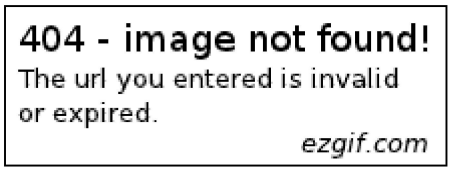
Speech recognition works through algorithms that analyze the sounds it hears by filtering it, digitiz
eventually come up with something that makes sense. Based on algorithms and all the inputs it ha
on what you are saying. However, it is more complicated than ever. Language dialects, accents an
noise, can play a role in whether the machine understands one or not. The machine can not see th

the sounds of cars in the background, as we humans can and therefore it is also important to inclu out all these background noise.

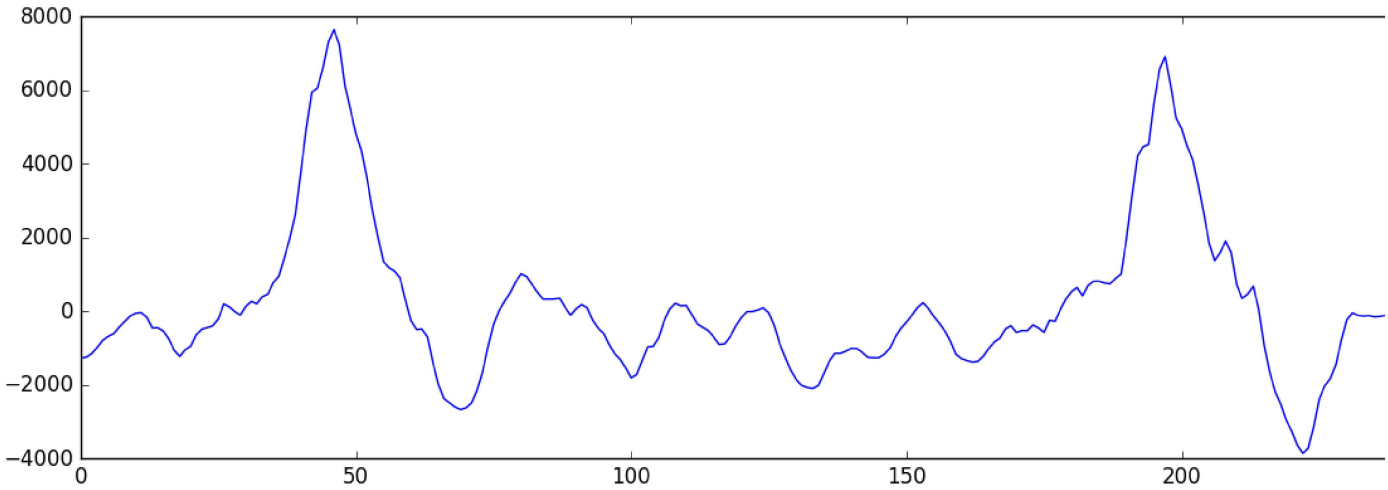
Sounds are represented as waves having 2 axes. Time on the x-axis and sound on the y-axis respe a value for the sound. The 'Librosa' package on Python allows you to load wav.files. This wav.files allows the software to train on a dataset with known spoken words or phrases and makes predicti hypotheses about what the user is saying.

Spectrograms

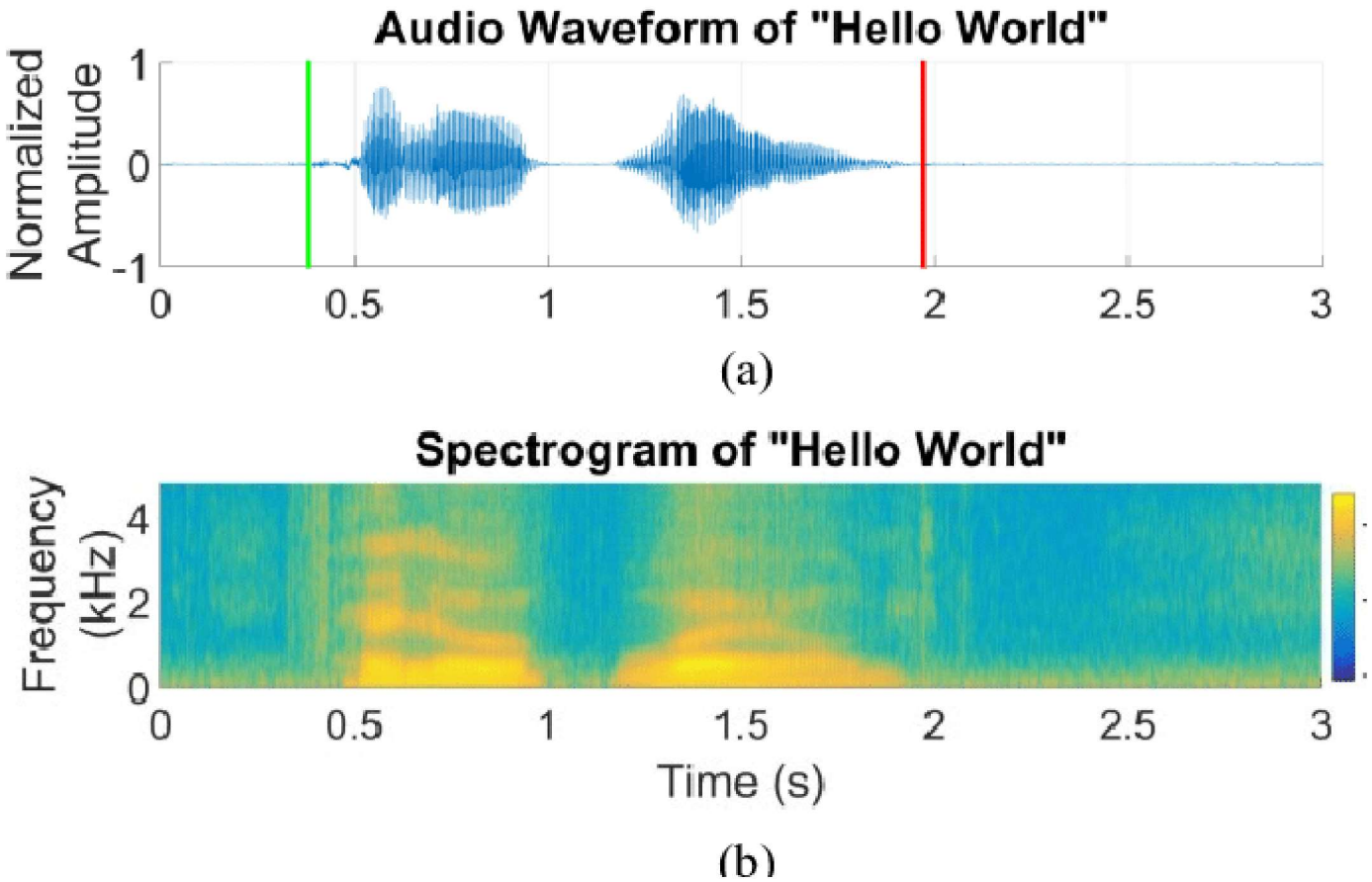
To analyze speech, the most common approach is by examining the sound waves of the actual sp every moment in time, they have a single value based on the height of the sound wave. The figure wave



To turn this sound wave into numbers, we just record of the height of the wave at equally-spaced p By sampling a certain sound or speech we record the number representing the height of the sounc This is shown below where an example of human speech is being presented



This recording is only 1/50th of a second long, yet it still shows how the frequency changes so dra these different frequencies combined make up human speech. It might seem like an impossible ta information, but through spectograms the machine can read the different frequencies over time ar is a visual representation of the spectrum of frequencies of a signal as it varies with time. This is s spectrogram is made based on the sentence: "Hello World".



The spectogram shows how the frequencies change drastically when the words are being spoken levels that help the machine detect the actual word. This is the theory most speech recognition is further by creating a speech recognition neural network.

```
from google.colab import drive
drive.mount('/mydrive')
```

```
Drive already mounted at /mydrive; to attempt to forcibly remount, call drive.mount("
```

Navigating to project directory

```
cd ..
```

```
/
```

```
cd mydrive/My Drive/speech_command_recognition
```

```
/mydrive/My Drive/speech_command_recognition
```

Installing required packages

```
!pip3 install http://download.pytorch.org/whl/cu80/torch-0.4.1-cp36-cp36m-linux_x86_64.whl
!pip3 install torchvision
!pip3 install librosa
!pip3 install Pillow==4.0.0
```

```
!pip3 install PIL
!pip3 install image
import PIL.Image
```

```
Collecting torch==0.4.1
  Downloading http://download.pytorch.org/whl/cu80/torch-0.4.1-cp36-cp36m-linux_x86_6
     |████████████████████████████████| 483.0MB 89.5MB/s
ERROR: torchvision 0.4.2 has requirement torch==1.3.1, but you'll have torch 0.4.1 wh
ERROR: fastai 1.0.59 has requirement torch>=1.0.0, but you'll have torch 0.4.1 which
Installing collected packages: torch
  Found existing installation: torch 1.3.1
    Uninstalling torch-1.3.1:
      Successfully uninstalled torch-1.3.1
Successfully installed torch-0.4.1
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from to
Collecting torch==1.3.1
  Downloading https://files.pythonhosted.org/packages/88/95/90e8c4c31cfc67248bf944ba4
     |████████████████████████████████| 734.6MB 23kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (fro
Installing collected packages: torch
  Found existing installation: torch 0.4.1
    Uninstalling torch-0.4.1:
      Successfully uninstalled torch-0.4.1
Successfully installed torch-1.3.1
Requirement already satisfied: librosa in /usr/local/lib/python3.6/dist-packages (0.6
Requirement already satisfied: six>=1.3 in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: decorator>=3.0.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in /usr/local/lib/python
Requirement already satisfied: resampy>=0.2.0 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: audioread>=2.0.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: joblib>=0.12 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: numba>=0.38.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: numpy>=1.8.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: llvmlite>=0.25.0dev0 in /usr/local/lib/python3.6/dist-
Collecting Pillow==4.0.0
  Downloading https://files.pythonhosted.org/packages/37/e8/b3fbf87b0188d22246678f8cd
     |████████████████████████████████| 5.6MB 2.7MB/s
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (fro
ERROR: torchvision 0.4.2 has requirement pillow>=4.1.1, but you'll have pillow 4.0.0
ERROR: scikit-image 0.15.0 has requirement pillow>=4.3.0, but you'll have pillow 4.0.
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have im
Installing collected packages: Pillow
  Found existing installation: Pillow 4.3.0
    Uninstalling Pillow-4.3.0:
      Successfully uninstalled Pillow-4.3.0
Successfully installed Pillow-4.0.0
ERROR: Could not find a version that satisfies the requirement PIL (from versions: no
ERROR: No matching distribution found for PIL
Requirement already satisfied: image in /usr/local/lib/python3.6/dist-packages (1.5.2
Requirement already satisfied: django in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: pytz in /usr/local/lib/python3.6/dist-packages (from d
Requirement already satisfied: sqlparse in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: olefile in /usr/local/lib/python3.6/dist-packages (fro
```

Importing required packages from project

```
import torch
from torchvision import models
import matplotlib.pyplot as plt
from data_loaders import data_loader
from trainer import train


PATH = 'datasets/speech_command_data/'
train_path = f'{PATH}train'
valid_path = f'{PATH}valid'
test_path = 'datasets/test'


train_dl, valid_dl, test_dl = data_loader.get_data_loaders(train_path=train_path, valid_pa
                                                           batch_size=64, num_workers=8)
```

```
Calculating mean and std of training dataset
Mean:tensor([0.2582, 0.1298, 0.3936])
Std:tensor([0.0526, 0.1985, 0.0859])
```

Resnet (Deep Residual Learning for Image Recognition)

This is a model type that exists predefined in the pytorch package, the package is distinctly differe an issue of a vanishing gradiant. The model can either be pretrained or not, if it is pretrained the m non pretrained model will not make use of predefined weights. The theory is that keeping the gradi ultimately improve accuracy of the model. There are advantages to using the pretrained model, mc not be standing intirely alone, however in many cases it is the non pretrained version has the poter restricted model.

```
model = models.resnet34(pretrained=False)
criterion = torch.nn.CrossEntropyLoss()


trainer = train.ModelTrainer(model=model, train_dl=train_dl, valid_dl=valid_dl, test_dl=te
```

```
Model moved to GPU
```

ModelTrainer modifies the fc layer so that last layer output corressponds to number of classes ir

```
trainer.model.fc
```

```
Linear(in_features=512, out_features=10, bias=True)
```

When you set a layer to "freeze" the parameter will no longer be trainable. If you do not change th part of the training. ModelTrainer freezes all layers and to prevent it from that, freezing must be freezing = FALSE.

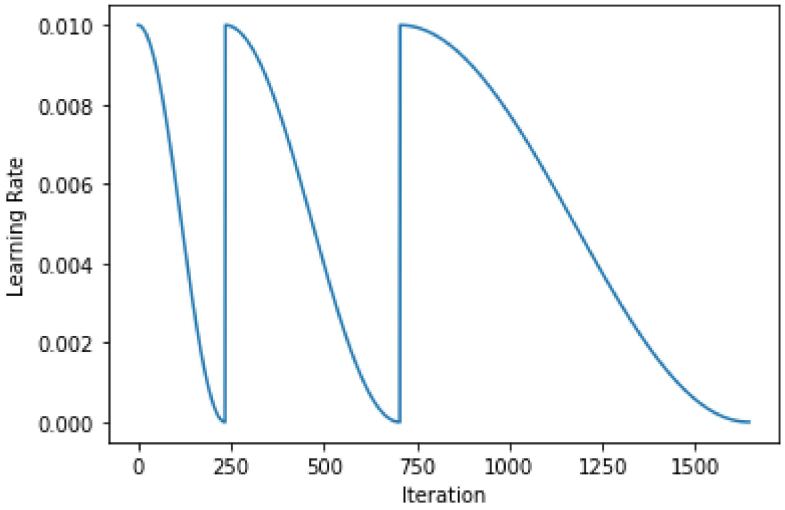
```
next(iter(trainer.model.parameters())).requires_grad
```

False

In Deep Learning we have a lot of parameters and all these parameters are linked to each other. W parameters and functions, we have an idea of what the output should be as we label our data. A m correct, which is why we have this "loss function". This measures the deviation from the actual val Descent as a method, you can update parameters and "search" for where the lowest point is. Stoch is a variant of learning speed cancellation that gradually reduces learning speed through training. I minimum and a very small change greatly increases the loss, then it is obviously not a good local r reset the learning speed and find a better local minimum.

```
trainer.dummy_sgdr(learning_rate=1e-2, n_cycles=3, cycle_len=1, cycle_mult=2)
```

```
/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:100: UserWarning:
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarni
```



Training the newly added fully connected layer for 3 epochs

Note:The fit method by default does SGDR. We have only passed value for n_cycles, by not passing default they both are 1), it is equivalent to training normally for 3 epochs

```
trainer.fit(n_cycles=3, learning_rate=1e-2)
```

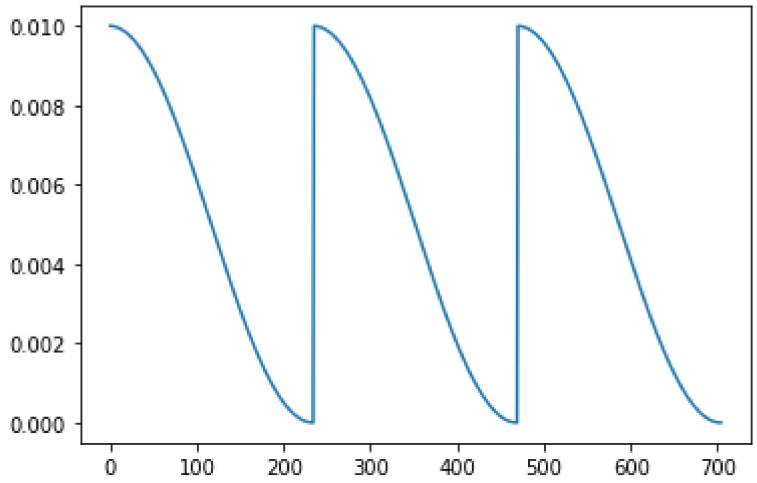
```
/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:100: UserWarning:
```

The fit method appends the learing rates at each iteration in a list named 'opt_lrs'

We can plot that list to visualize the graph of learning rate vs iterations for current SGDR

```
plt.plot(trainer.opt_lrs)
```

```
[<matplotlib.lines.Line2D at 0x7f4a7b5af550>]
```



Each time the learning rate drops to the minimum point, which in this case is every 100 iterations, learning rate is decreases. This allows you to see where the local minimum is. If the local minimur the cycle will be at the same local minimum unless it converges it to another minimum.

Unfreezing all the layers to finetune the network

```
trainer.unfreeze()
```

Verifying that the model is unfreezed

```
next(iter(trainer.model.parameters())).requires_grad
```

```
True
```

Training the model using SGDR for 4 cycles with cycle_mult=2

The number of epochs between resetting the learning rate is set at cycle_len, and the number of ti cycles. Cycle_mult multiplies the cycle length after each cycle. In this case the cyclen_len is 1 epo multiplied by cycle_len to increase the cycle lenght. This is the 'restart' in 'Stochastic Gradient Des

SGDR is a good way cause once we get closer enough to the optimal and stabil area, the resetting beter. Therefor SGDR is better than create "ensembles and try different starting point.

```
trainer.fit(n_cycles=4, cycle_len=1, cycle_mult=2, learning_rate=1e-2, save_snapshot=True)
```

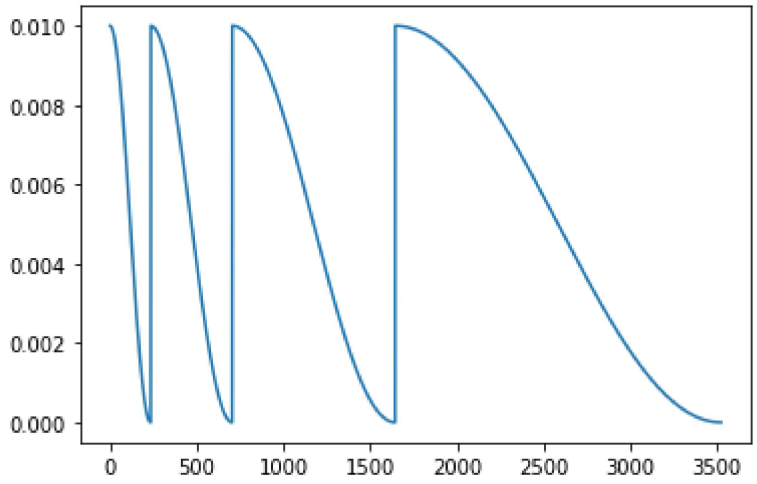
```
Cycle:1
        Epoch:1/1
/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:100: UserWarning:
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarni
        Train Loss:0.73702472448349
        Valid Loss:0.00408627699688077 Valid Accuracy:0.912
        ---------------------------------------------------
        Snapshot saved!
Cycle:2
        Epoch:1/2
        Train Loss:0.3187691569328308
        Valid Loss:0.0034463248029351235 Valid Accuracy:0.9275
        ---------------------------------------------------
        Epoch:2/2
        Train Loss:0.12158048897981644
        Valid Loss:0.0023331936029717328 Valid Accuracy:0.9495
        ---------------------------------------------------
        Snapshot saved!
Cycle:3
        Epoch:1/4
        Train Loss:0.09858879446983337
        Valid Loss:0.005790891833603382 Valid Accuracy:0.886
        ---------------------------------------------------
        Epoch:2/4
        Train Loss:0.08263754844665527
        Valid Loss:0.002646792609244585 Valid Accuracy:0.9435
        ---------------------------------------------------
        Epoch:3/4
        Train Loss:0.0017026265850290656
        Valid Loss:0.001969364510849118 Valid Accuracy:0.9625
        ---------------------------------------------------
        Epoch:4/4
        Train Loss:0.004017611499875784
        Valid Loss:0.0019368388038128615 Valid Accuracy:0.9615
        ---------------------------------------------------
        Snapshot saved!
Cycle:4
        Epoch:1/8
        Train Loss:0.004596511367708445
        Valid Loss:0.005022370047867298 Valid Accuracy:0.925
        ---------------------------------------------------
        Epoch:2/8
        Train Loss:0.029341837391257286
        Valid Loss:0.004830752152949571 Valid Accuracy:0.924
        ---------------------------------------------------
        Epoch:3/8
        Train Loss:0.01854495145380497
        Valid Loss:0.002509289937093854 Valid Accuracy:0.9595
        ---------------------------------------------------
        Epoch:4/8
        Train Loss:0.001958489418029785
        Valid Loss:0.00222941267862916 Valid Accuracy:0.9550000000000001
        ---------------------------------------------------
        Epoch:5/8
        Train Loss:0.001399437547661364
        Valid Loss:0.0020779413301497698 Valid Accuracy:0.9625
        ---------------------------------------------------
        Epoch:6/8
        Train Loss:0.004617214202880859
        Valid Loss:0.002069487031549215 Valid Accuracy:0.963
        ---------------------------------------------------
```

```
Epoch:7/8
Train Loss:0.0007830460672266781
Valid Loss:0.002160998033359647 Valid Accuracy:0.9590000000000001
---------------------------------------------------
Epoch:8/8
Train Loss:0.0006763140554539859
Valid Loss:0.00214707931317389 Valid Accuracy:0.96
---------------------------------------------------
Snapshot saved!
```

Again, plotting the opt_lrs list to visualize SGDR

```
plt.plot(trainer.opt_lrs)
```

```
[<matplotlib.lines.Line2D at 0x7f4a6af37898>]
```



Predicting test set with top 3 snapshots (models)

Snapshots are a way to train more than one model. The models provide different predictions for t the wrong prediction then it may be the other gives the right one. With snapshots, we capture / ta local minimum. Once it has been through the entire model, we get a neural network. This approa biases and gather for networks is called Snapshot Ensembles. In this case we predict test set wit

```
snapshot_test_acc_3 = trainer.predict('saved/2018-09-24_15:30:53', n_models=3)
```

```
print(f'Snapshot Ensemble Test Accuracy with top 3 models:{snapshot_test_acc_3}')
```

```
Snapshot Ensemble Test Accuracy with top 3 models:0.9042553191489362
```

Conclusion:

Overall the model is quite satisfying with an accuracy of 90%, but bear in mind this is only made or insane and impressive work that is required to create a more advanced speach recognition.

- Final notes:

The spectogram files were collected in a different colab. The creation of the spectograms have be took serveral hours. If it was included in this tutorial the whole Colab would have required many hc for a tutorial.