# Vaihe 1, Setup:

import os -Lisää kirjaston, sen kautta voidaan hakea tietoja tiedostojärjestelmästä.  
import pathlib -Se lisää kirjaston, joka mahdollistaa käyttöjärjestelmäriippumattomat tiedostoviittaukset.  
  
import matplotlib.pyplot as plt -Sisältää monipuoliset työkalut erilaisten kuvaajien ja kaavojen tekemiseen

.   
import numpy as np -Sisältää numeerisen laskennan kannalta keskeisiä työkaluja: ndarray

import seaborn as sns – sama kuin matplotlib

import tensorflow as tf –  tarjoaa työkaluja, joiden avulla voidaan rakentaa **syvien hermo- /neuroverkkojen malleja( se käyttää symbolista matematiikkaa, tietovirtaa, ja differentoituvaa ohjelmointia.)**   
  
from tensorflow.keras import layers – sen avulla luodaan erilaisia ​​kerroksia/pintoja, joista jokaisella on oma tehtävänsä hermoverkossa

from tensorflow.keras import models - ???

from IPython import display - ???  
  
# Set the seed value for experiment reproducibility.  
Seed = 42 – asettaa seed muuttujan arvoksi 42  
tf.random.set\_seed(seed) –asettaa globaalin seedin, jolla varmistetaan toistettavuus   
np.random.seed(seed) - asettaa randomin seedin arvoksi 42

# Vaihe 2, Import the mini-speech commands dataset:

DATASET\_PATH = 'data/mini\_speech\_commands' – asettaa DATASET\_PATH muuttujan arvoksi ‘data/mini\_speech\_commands’   
  
data\_dir = pathlib.Path(DATASET\_PATH) - se asettaa Path-olion, joka on sama kuin DATASET\_PATH, mikä mahdollistaa käyttöjärjestelmästä riippumattoman tavan käsitellä tätä polkua ja siinä olevia tiedostoja.

if not data\_dir.exists(): - if-lause, joka tarkistaa onko tiedosto on olemassa, jos ei ole, niin se lataa ja purkaa sen.  
  tf.keras.utils.get\_file(   
      'mini\_speech\_commands.zip',  
      origin="http://storage.googleapis.com/download.tensorflow.org/data/mini\_speech\_commands.zip",  
      extract=True,  
      cache\_dir='.', cache\_subdir='data')

commands = np.array(tf.io.gfile.listdir(str(data\_dir))) - Hakee tiedostojen ja kansiojen nimet data\_dir hakemistosta ja muuntaa niitä numpy-taulukoksi.

commands = commands[(commands != 'README.md') & (commands != '.DS\_Store')] – suodattaa **commands-**taulukosta pois rivit, joissa arvona on joko **‘README.md’** tai **‘.DS\_Store’.**

print('Commands:', commands) -tulostaa taulukon.

# Vaihe 3:

train\_ds, val\_ds = tf.keras.utils.audio\_dataset\_from\_directory(  
    directory=data\_dir,  
    batch\_size=64,  
    validation\_split=0.2,  
    seed=0,  
    output\_sequence\_length=16000,  
    subset='both')

- luo **kaksi TensorFlow-datasettiä**, train\_ds ja val\_ds, äänitiedostoista, jotka sijaitsevat annetussa hakemistossa data\_dir, Tämä tapahtuu käyttämällä tf.keras.utils.audio\_dataset\_from\_directory -komentoa.

train\_ds sisältää koulutusaineiston, ja val\_ds sisältää validointiaineiston.

batch\_size määrittää kuinka monta äänitiedostoa kussakin erässä käytetään.

validation\_split=0.2 jakaa aineiston osaksi koulutus- ja validointiaineistoa. Tässä tapauksessa 20% äänitiedostoista varataan validointiin.

output\_sequence\_length=16000 määrittää äänisekvenssin pituuden, tässä tapauksessa 16000 näytettä. subset=’both’ tarkoittaa, että sekä koulutus- että validointiaineistot luodaan

label\_names = np.array(train\_ds.class\_names) -luo taulukon nimeltä label\_names joka sisältää luokkien nimet train\_ds tietojoukossa.

Print() -tuolostaa, jotain, ehkä se on välilyönti   
print("label names:", label\_names) -tulostaa label\_names,

# Vaihe 4:

train\_ds.element.spec() -hakee train\_ds-tietovaraston ensimmäisen elementin erittelyn.

def squeeze(audio, labels):  
  audio = tf.squeeze(audio, axis=-1)  
  return audio, labels

- puristaa taulukon yhden ulottuvuuden pois, koska datasetti sisältää vain 1 kanavan. Tässä tapauksessa funktio ottaa **audio**- ja **labels**-parametrit, ja se puristaa **audio**-taulukon viimeisen ulottuvuuden, jos sen koko on 1. Tämä auttaa yksinkertaisemaan taulukkoa.

train\_ds = train\_ds.map(squeeze, tf.data.AUTOTUNE)-tämä koodi käyttää squeeze funktiota, ja tf.data.AUTOTUNE toimintoa joka mahdollistaa **dynaamisen suorituskyvyn optimoinnin** tietojoukkojen käsittelyssä

val\_ds = val\_ds.map(squeeze, tf.data.AUTOTUNE)- sama kuin viimeinen

# Vaihe 5:

test\_ds = val\_ds.shard(num\_shards=2, index=0) – tässä vaiheessa perustetaan uusi datasetti val\_ds datasetistä, jota käytetään neuroverkon testaamiseen. Eli se ottaa puolet val\_ds datasetistä.

val\_ds = val\_ds.shard(num\_shards=2, index=1) – sama kuin viimeinen.

# Vaihe 6:

for example\_audio, example\_labels in train\_ds.take(1):    
  print(example\_audio.shape)  
  print(example\_labels.shape)

1. For-looppi, alustaa muuttujat example\_audio ja example\_labels ensimmäisellä esimerkillä koulutusaineistosta, sitten tulostaa example\_audio muuttujan muodon ja example\_labels muttujan muodon.

label\_names[[1,1,3,0]] - Se viittaa merkkijonoon label\_names[[1,1,3,0]]

# Vaihe 7:

plt.figure(figsize=(16, 10)) - Luo uuden kuvaajan koon ollessa 16x10 tuuma.

rows = 3 - Määrittää rivien määrän kuvaajassa oleville

alikuvioille.

cols = 3 - Määrittää sarakkeiden määrän kuvaajassa oleville

alikuvioille.

n = rows \* cols - Laskee alikuvioita, yhteensä 9 kappaletta.

for i in range(n): - For-looppi, joka käy läpi alikuvioita

plt.subplot(rows, cols, i+1) - Luo alikuvion ja asettaa sen sijainnin rivillä

ja sarakkeessa.

audio\_signal = example\_audio[i] - Hakee äänisignaalin example\_audio:sta indeksillä

i

plt.plot(audio\_signal) - Tulostaa kuvaajan

plt.title(label\_names[example\_labels[i]]) - Asettaa alikuvion otsikon

esimerkin luokan nimen

perusteella.

  plt.yticks(np.arange(-1.2, 1.2, 0.2)) - Asettaa y-akselin merkinnät,

  plt.ylim([-1.1, 1.1] - Asettaa y-akselin näyttöalueen välille -1.1 ja 1.1.

# Vaihe 8, Convert waveforms to spectrograms with STFT:

def get\_spectrogram(waveform): - Määrittelee funktion, nimeltään get\_spectrogram, jossa

paramterinä toimii waveform   
  spectrogram = tf.signal.stft(waveform, frame\_length=255, frame\_step=128)

- Laskee ääniaallon lyhytaikaisen Fourier-muunnoksen (STFT) spektrogrammin. Käytetään 255 näytettä frame\_length, joissa on 128 steppiä (???näytteen päällekkäisyys???)

spectrogram = tf.abs(spectrogram) - Ottaa spektrogrammin absoluuttisen arvon, joka kuvaa äänen

voimakkuutta taajuusalueittain.

spectrogram = spectrogram[..., tf.newaxis] - Lisää yhden ulottuvuuden spektrogrammiin, jotta sitä voidaan

käsitellä helpommin.

return spectrogram -  Palauttaa tuloksen, eli ääniaallon spektrogrammin.

# Vaihe 9:

for i in range(3): -  Käy läpi kolme kertaa.

label = label\_names[example\_labels[i]] - Hakee label\_names-listasta indeksin i mukaisen nimen ja

tallentaa sen muuttujaan label.

waveform = example\_audio[i] - Ottaa example\_audio-listasta indeksin i mukaisen ääniaallon ja

tallentaa sen muuttujaan waveform.

  spectrogram = get\_spectrogram(waveform) - Kutsuu get\_spectrogram-funktiota ääniaallon waveform

perusteella ja tallentaa tuloksen muuttujaan spectrogram.  
  
  print('Label:', label) - tulostaa label

print('Waveform shape:', waveform.shape) - ääniaallon muodon (shape)

  print('Spectrogram shape:', spectrogram.shape)  - spektrogrammin muodon (shape)

  print('Audio playback') - Tulostaa tekstin hakasulkeista

  display.display(display.Audio(waveform, rate=16000)) – Äänen toisto

# Vaihe 10, Define a function for displaying spectrogram:

def plot\_spectrogram(spectrogram, ax): - Määrittelee funktion nimeltä plot\_spectrogram, joka ottaa

parametreikseen spectrogram-matriisin ja ax-akselin  
  if len(spectrogram.shape) > 2: - Tarkistaa, onko spectrogram-matriisin ulottuvuus suurempi kuin 2.

    assert len(spectrogram.shape) == 3 - Varmistaa, että spectrogram-matriisin ulottuvuus on tarkalleen 3.

    spectrogram = np.squeeze(spectrogram, axis=-1) - Poistaa yhden ulottuvuuden spectrogram-matriisista.

  log\_spec = np.log(spectrogram.T + np.finfo(float).eps) - Laskee logaritmin spectrogram-matriisin

transponoidusta versiosta.

  height = log\_spec.shape[0] - Määrittää korkeuden log\_spec-matriisille.

  width = log\_spec.shape[1] - Määrittää leveyden log\_spec-matriisille.

  X = np.linspace(0, np.size(spectrogram), num=width, dtype=int) - Luo aikavälin X spectrogram-matriisin

leveyden perusteella.

  Y = range(height) - Luo korkeusarvot Y log\_spec-matriisin

korkeuden perusteella.

  ax.pcolormesh(X, Y, log\_spec) - Piirtää spektrogrammin X-Y -akselille

# Vaihe 11, Luo kaksi kuvaajaa, Waveform ja Spectrogram:

fig, axes = plt.subplots(2, figsize=(12, 8)) - Luo kaksi kuvaajaa (subplots) ja määrittää niiden koon.

timescale = np.arange(waveform.shape[0]) - Luo aikaskaalan, joka vastaa waveform-muuttujan pituutta.

axes[0].plot(timescale, waveform.numpy()) - Piirtää ensimmäisen kuvaajan (Waveform) aikaskaalalla ja

waveform-datalla.

axes[0].set\_title('Waveform') - Asettaa ensimmäisen kuvaajan otsikon.

axes[0].set\_xlim([0, 16000]) - Asettaa ensimmäisen kuvaajan x-akselin rajat.   
  
plot\_spectrogram(spectrogram.numpy(), axes[1]) - Piirtää toisen kuvaajan (Spectrogram) käyttäen

spectrogram-dataa.

axes[1].set\_title('Spectrogram') - Asettaa toisen kuvaajan otsikon.

plt.suptitle(label.title()) - Asettaa yleisen otsikon koko kuvaajalle.

plt.show() - Tulostaa kuvaajat

# Vaihe 12, Luodaan spectrogram datasettiä audio datasetistä:

def make\_spec\_ds(ds): - Määrittelee funktion nimeltä make\_spec, funktio ottaa parametreikseen

ds datan.

  return ds.map( - Palauttaa tuloksen, soveltaa annettua map\_func jokaiseen audio,label-pariin.

      map\_func=lambda audio,label: (get\_spectrogram(audio), label) - map\_func ottaa audio- äänisignaalin ja label-luokan ja palauttaa uuden parin (get\_spectrogram(audio), label).  Get...- luo äänisignaalista spektrogrammin.

      num\_parallel\_calls=tf.data.AUTOTUNE) – Määrittää, että suoritettavien kutsujen määrä optimoidaan

automaattisesti.

# Vaihe 13, Koulutusdata, validiointidata ja testidata:

train\_spectrogram\_ds = make\_spec\_ds(train\_ds) - Tämä rivi luo koulutusdatan spektrogrammidatasetin (train\_spectrogram\_ds) koulutusdatan (train\_ds) perusteella.

val\_spectrogram\_ds = make\_spec\_ds(val\_ds) - Tämä rivi luo validointidatan spektrogrammidatasetin

(val\_spectrogram\_ds) validointidatan (val\_ds) perusteella. Sitä käytetään koulutuksen aikana.

test\_spectrogram\_ds = make\_spec\_ds(test\_ds) - Tämä rivi luo testidatan spektrogrammidatasetin (test\_spectrogram\_ds) testidatan (test\_ds) perusteella.

Käytetään koulutuksen jälkeen.

# Vaihe 14:

for example\_spectrograms, example\_spect\_labels in train\_spectrogram\_ds.take(1):

break

Ottaa ensimmäisen alkion train\_spectrogram\_ds -tietojoukosta ja lopettaa sen jälkeen suorituksen

# Vaihe 15, uusi plot, again...:

Rows = 3 - Määrittää rivit avroksi 3.  
cols = 3 - Määrittää sarakkeet arvoksi 3, saamme 3x3 ruudukon.   
n = rows\*cols -Laskee kuvien määrä, joka on yhteeensä 9 kuvaa.   
fig, axes = plt.subplots(rows, cols, figsize=(16, 9)) - Luodaan kuva fig ja 9 alikuvaa axes annetulla

kokoisella.  
  
for i in range(n): - For-looppi käy läpi 9 kuvaa, i-arvot 0-8   
    r = i // cols - Lasketaan rivin indeksit kuvan sijainnin määrittämiseksi  
    c = i % cols - Lasketaan sarakkeen indeksit.   
    ax = axes[r][c] - Asettaa akselit   
    plot\_spectrogram(example\_spectrograms[i].numpy(), ax) -  Piirretään spektrogrammi   
    ax.set\_title(label\_names[example\_spect\_labels[i].numpy()]) - Asetetaan alikuvan otsikko   
  
plt.show() - Tulostaa kuvan

# Vaihe 16, Mallin rakentaminen ja kouluttaminen. Optimoidaan koulutuksen, testauksen ja validoinnin:

train\_spectrogram\_ds = train\_spectrogram\_ds.cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)

1. cache() - Tämä välimuistittaa koulutusdatan, jotta esikäsittelytoimenpiteitä ei tarvitse toistaa joka kerta uudelleen.
2. **Shuffle(10000)** - Sekoittaa näytteet satunnaisessa järjestyksessä, jotta verkkoon syötettävät näytteet ovat aina eri järjestyksessä.
3. **prefetch(tf.data.AUTOTUNE)** - Lataa datan etukäteen, jotta se on valmiina seuraavaa koulutusaskelta vartenAUTOTUNE säätää dynaamisesti latausmäärää suorituksen optimoimiseksi

val\_spectrogram\_ds = val\_spectrogram\_ds.cache().prefetch(tf.data.AUTOTUNE)

1. Sama kuin viiminen, mutta ilman shuffle toimintoa.

test\_spectrogram\_ds = test\_spectrogram\_ds.cache().prefetch(tf.data.AUTOTUNE)

# Vaihe 17:

input\_shape = example\_spectrograms.shape[1:] - Määrittelee input\_shape-muuttujan arvoksi example\_spectrograms-taulukon muodon (shape) indeksistä 1 eteenpäin. Tämä tarkoittaa, että input\_shape saa kaikki ulottuvuudet paitsi ensimmäisen.

print('Input shape:', input\_shape) - Tulostaa näytölle tekstin “Input shape:” ja sen jälkeen input\_shape-muuttujan arvon.

num\_labels = len(label\_names) - Laskee label\_names-taulukon alkioiden määrän ja tallentaa sen num\_labels-muuttujaan.  
.  
norm\_layer = layers.Normalization() - Luo uuden normalisointikerroksen (Normalization) ja tallentaa sen norm\_layer-muuttujaan.

# Vaihe 18:

norm\_layer.adapt(data=train\_spectrogram\_ds.map(map\_func=lambda spec, label: spec)) - Tämä rivi sovittaa normalisointikerroksen (norm\_layer) koulutusdatan spektrogrammeihin. Se käyttää map\_func, joka ottaa spektrin ja merkinnän parametreina ja palauttaa spektrin.  
  
model = models.Sequential([  
    layers.Input(shape=input\_shape),  
    layers.Resizing(32, 32),  
    norm\_layer,  
    layers.Conv2D(32, 3, activation='relu'),  
    layers.Conv2D(64, 3, activation='relu'),  
    layers.MaxPooling2D(),  
    layers.Dropout(0.25),  
    layers.Flatten(),  
    layers.Dense(128, activation='relu'),  
    layers.Dropout(0.5),  
    layers.Dense(num\_labels),  
])

1. Tämä luo mallin käyttäen Kerasin Sequential-mallia. Malli koostuu useista kerroksista, jotka on määritelty listassa. Näihin kuuluvat syötekerros (layers.Input), koonmuutoskerros (layers.Resizing), normalisointikerros (norm\_layer), kaksi konvoluutiokerrosta (layers.Conv2D), maksimipoolauskerros (layers.MaxPooling2D), pudotuskerros (layers.Dropout), tasauskerros (layers.Flatten) ja kaksi Dense kerrosta (layers.Dense). Viimeinen Dense kerros ei ole määritelty aktivaatiofunktiolla, joten se tuottaa raakalähtöarvot.  
     
   model.summary() - Tämä tulostaa yhteenvedon luodusta mallista, mukaan lukien kerrosten tyyppi, koko ja parametrit.

# Vaihe 19:

model.compile( - Tämä metodi määrittää mallin kääntämisen vaiheet.  
    optimizer=tf.keras.optimizers.Adam(), - Tämä asettaa mallin optimointialgoritmin. Täsä tapauksessa käytetään Adam-optimointialgoritmia, joka on tehokas gradienttimenetelmä.  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), - Tämä määrittää tappiofunktion. Tässä tapauksessa käytetään harvan kategorisen ristientropian menetelmää, joka soveltuu luokitteluongelmiin.   
    metrics=['accuracy'], - Tämä määrittää mallin suorituskyvyn mittarit. Tässä tapauksessa käytämme tarkkuutta.  
)

# Vaihe 20:

EPOCHS = 10 - Asettaa EPOCHS-muuttujan arvoksi 10. Tämä määrittää, kuinka monta kertaa malli käy läpi koulutusdatan.  
history = model.fit(  
    train\_spectrogram\_ds,  
    validation\_data=val\_spectrogram\_ds,  
    epochs=EPOCHS,  
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1, patience=2),

1. Käynnistää mallin koulutuksen käyttäen train\_spectrogram\_ds-koulutusdataa ja val\_spectrogram\_ds-validointidataa. Koulutus suoritetaan EPOCHS-kierroksen ajan. Lisäksi käytetään EarlyStopping-kutsua, joka keskeyttää koulutuksen, jos validointitulos ei parane patience-kierroksen aikana.  
   )

# Vaihe 21:

metrics = history.history - Tallentaa history-objektin metrics-muuttujaan.

plt.figure(figsize=(16,6)) - Luo kuvaajan, jonka koko on 16x6 tuumaa.

plt.subplot(1,2,1) - Luo alikuvion, joka on ensimmäinen (vasen) osa kahdesta.

plt.plot(history.epoch, metrics['loss'], metrics['val\_loss']) - Piirtää käyrän, joka esittää menetystä (loss) ja validointimenetystä (val\_loss) ajan funktiona.

plt.legend(['loss', 'val\_loss']) - Näyttää selitteen käyrien värien vastaavuudesta

plt.ylim([0, max(plt.ylim())]) - Asettaa y-akselin rajoiksi 0 ja suurimman arvon, joka on tällä hetkellä näkyvissä.

plt.xlabel('Epoch') - Asettaa x-akselin otsikon “Epoch”

plt.ylabel('Loss [CrossEntropy]') - Asettaa y-akselin otsikon “Loss [CrossEntropy]”.  
  
plt.subplot(1,2,2) - Luo toisen alikuvion, joka on oikea osa kahdesta.

plt.plot(history.epoch, 100\*np.array(metrics['accuracy']), 100\*np.array(metrics['val\_accuracy'])) - Piirtää käyrät, jotka esittävät tarkkuutta (accuracy) ja validointitarkkuutta (val\_accuracy) prosentteina ajan funktiona.

plt.legend(['accuracy', 'val\_accuracy']) - Näyttää selitteen käyrien värien vastaavuudesta.

plt.ylim([0, 100]) - Asettaa y-akselin rajoiksi 0 ja 100

plt.xlabel('Epoch') - Asettaa x-akselin otsikon “Epoch”

plt.ylabel('Accuracy [%]') - Asettaa x-akselin otsikon “Accuracy [%]”

# Vaihe 22:

model.evaluate(test\_spectrogram\_ds, return\_dict=True) - Tämä koodirivi suorittaa mallin arvioinnin testijoukolla ja palauttaa tulokset sanakirjana, joka sisältää menetyksen ja tarkkuuden

# Vaihe 23:

y\_pred = model.predict(test\_spectrogram\_ds) - Mallili ennustaa testispektrogrammeille ja tallentaa ennusteet y\_pred-muuttujaan.

y\_pred = tf.argmax(y\_pred, axis=1) - Valitsee ennustetuista luokista korkeimman arvon ja tallentaa sen y\_pred-muuttujaan.

y\_true = tf.concat(list(test\_spectrogram\_ds.map(lambda s,lab: lab)), axis=0) - Yhdistää testispektrogrammien todelliset luokat y\_true-muuttujaan.

confusion\_mtx = tf.math.confusion\_matrix(y\_true, y\_pred) - Laskee sekaannusmatriisin ennustettujen ja todellisten luokkien välillä.  
plt.figure(figsize=(10, 8)) - Piirtää lämpökartan sekaannusmatriisista, jossa x-akselilla on ennustetut arvot ja y-akselilla todelliset arvot.

sns.heatmap(confusion\_mtx,  
            xticklabels=label\_names,  
            yticklabels=label\_names,  
            annot=True, fmt='g') - Näyttää kartan, jossa on ennusteen ja todellisen luokan merkinnät  
plt.xlabel('Prediction') - Asettaa x-akselin otsikon “Prediction”  
plt.ylabel('Label') - Asettaa x-akselin otsikon “Label”  
plt.show() - Tulostaa ylös

# Vaihe 24:

x = data\_dir/'no/01bb6a2a\_nohash\_0.wav' - Määrittää tiedostonimen “no/01bb6a2a\_nohash\_0.wav” muuttujalle x.

x = tf.io.read\_file(str(x)) - Lukee tiedoston sisällön ja tallentaa sen muuttujaan x.

x, sample\_rate = tf.audio.decode\_wav(x, desired\_channels=1, desired\_samples=16000,) - Dekoodaa äänitiedoston x ja määrittää näytteiden määräksi 16000.

x = tf.squeeze(x, axis=-1) - Poistaa yhden ulottuvuuden x:stä.  
waveform = x - Asettaa waveform-muuttujan arvoksi x.  
x = get\_spectrogram(x) - Laskee spektrogrammin x:stä.  
x = x[tf.newaxis,…] - Lisää yhden ulottuvuuden x:ään.  
  
prediction = model(x) - Ennustaa mallilla model luokkia x:n perusteella.  
x\_labels = ['no', 'yes', 'down', 'go', 'left', 'up', 'right', 'stop'] - Määrittää luokkien nimet.  
plt.bar(x\_labels, tf.nn.softmax(prediction[0])) - Luo pylväskaavion ennustetuista todennäköisyyksistä.  
plt.title('No') - Asettaa kaavion otsikoksi “No”.  
plt.show() - Tulostaa kaavion.  
  
display.display(display.Audio(waveform, rate=16000)) -  Näyttää ääniaaltomuodon.

# Vaihe 25:

class ExportModel(tf.Module): - Määrittelee uuden luokan nimeltä ExportModel, joka perii tf.Module-luokan ominaisuudet.

  def \_\_init\_\_(self, model): - Luokan konstruktori, joka ottaa parametrina model-olion ja tallentaa sen self.model-muuttujaan.

    self.model = model  
  
     
    self.\_\_call\_\_.get\_concrete\_function(  
        x=tf.TensorSpec(shape=(), dtype=tf.string)) - Hakee konkreettisen funktion call-metodille, joka ottaa syötteenä yhden merkkijonotensorin.  
    self.\_\_call\_\_.get\_concrete\_function(  
       x=tf.TensorSpec(shape=[None, 16000], dtype=tf.float32)) - Hakee konkreettisen funktion call-metodille, joka ottaa syötteenä liukulukutensorin, jonka muoto on [None, 16000]  
  
  
  @tf.function - Koristefunktio, joka muuntaa call-metodin TensorFlow-graafiksi, jotta se voidaan suorittaa tehokkaasti.  
  def \_\_call\_\_(self, x):

1. def \_\_call\_\_(self, x):: call-metodi, joka ottaa syötteenä x-tensorin.
2. Jos x on merkkijonotensori, lataa tiedoston ja dekoodaa sen.
3. Muuntaa dekoodatun äänitiedoston spektrogrammiksi.
4. Suorittaa mallin ennusteen.
5. Palauttaa ennusteet, luokkatunnisteet ja luokkanimet.

# If they pass a string, load the file and decode it.   
    if x.dtype == tf.string:  
      x = tf.io.read\_file(x)  
      x, \_ = tf.audio.decode\_wav(x, desired\_channels=1, desired\_samples=16000,)  
      x = tf.squeeze(x, axis=-1)  
      x = x[tf.newaxis, :]  
  
    x = get\_spectrogram(x)    
    result = self.model(x, training=False)  
  
    class\_ids = tf.argmax(result, axis=-1)  
    class\_names = tf.gather(label\_names, class\_ids)  
    return {'predictions':result,  
            'class\_ids': class\_ids,  
            'class\_names': class\_names}

# Vaihe 26:

export = ExportModel(model) - Tässä luodaan uusi muuttuja nimeltä “export”, joka saa arvokseen “model”-nimisen mallin. Tämä rivi liittyy mallin vientiin tai tallentamiseen.

export(tf.constant(str(data\_dir/'no/01bb6a2a\_nohash\_0.wav'))) - Tässä koodissa “export”-muuttujaan syötetään äänitiedosto, jonka polku on “data\_dir/no/01bb6a2a\_nohash\_0.wav”. Tämä voi liittyä mallin ennustamiseen äänitiedostosta.

tf.saved\_model.save(export, "saved") - Tässä malli tallennetaan nimellä “saved”. Tämä rivi liittyy mallin tallentamiseen.  
imported = tf.saved\_model.load("saved") - Tässä ladataan aiemmin tallennettu malli nimellä “saved”. Tämä voi liittyä mallin uudelleenkäyttöön tai jatkokehitykseen.   
imported(waveform[tf.newaxis, :]) - Tässä käytetään ladattua mallia syöttämällä sille “waveform”-tieto uudessa akselissa. Tämä voi liittyä mallin käyttämiseen ennustamiseen tai analyysiin