# DLAP_4

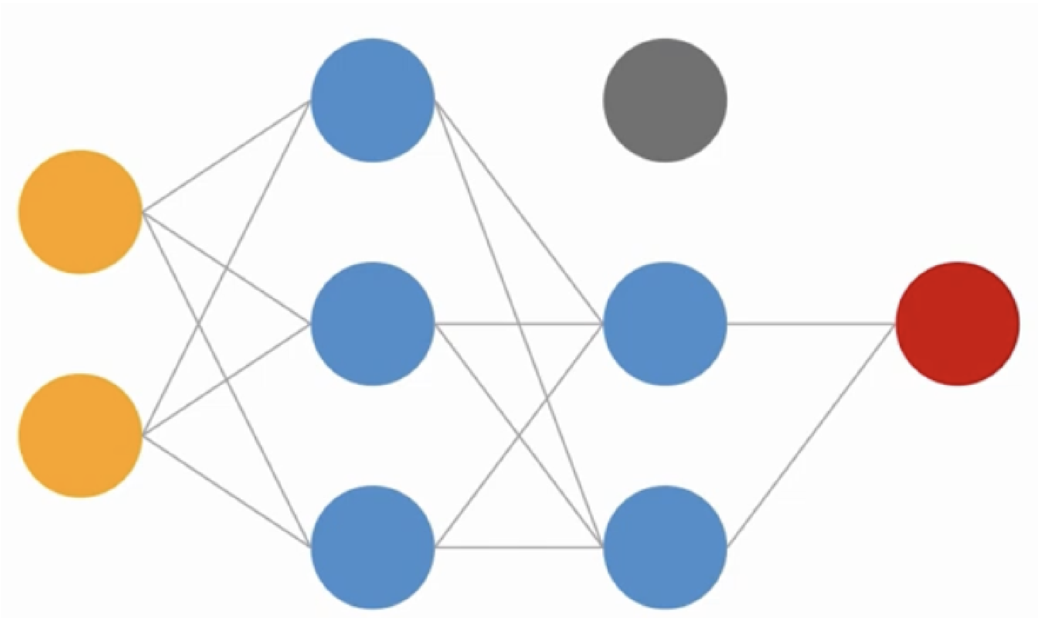
mlp_genre_classifier.py

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras

# path to json file that stores MFCCs and genre labels for each processed segment
DATA_PATH = "path/to/dataset/in/json/file"

def load_data(data_path):
    """Loads training dataset from json file.
        :param data_path (str): Path to json file containing data
        :return X (ndarray): Inputs
        :return y (ndarray): Targets
    """

    with open(data_path, "r") as fp:
        data = json.load(fp)

    # convert lists to numpy arrays
    X = np.array(data["mfcc"])
    y = np.array(data["labels"])

    print("Data succesfully loaded!")

    return  X, y


if __name__ == "__main__":

    # load data
    X, y = load_data(DATA_PATH)

    # create train/test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    # build network topology
    model = keras.Sequential([

        # input layer
        keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])),

        # 1st dense layer
        keras.layers.Dense(512, activation='relu'),

        # 2nd dense layer
        keras.layers.Dense(256, activation='relu'),

        # 3rd dense layer
        keras.layers.Dense(64, activation='relu'),

        # output layer
        keras.layers.Dense(10, activation='softmax')
    ])

    # compile model
    optimiser = keras.optimizers.Adam(learning_rate=0.0001)
    model.compile(optimizer=optimiser,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()

    # train model
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=32, epochs=50)
```

Solving overfitting

- 더 간단한 구조

    - 레이어 제거
    - 파라미터 줄이기
- 데이터 증강
    - pitch shifting
    - time stretching
    - Adding background noise
    - …
- Early stopping
- Dropout
    - 배치마다 랜덤하게 뉴런을 제거
    - robust하게 만듦



- Regularization
    - error function에 penalty를 줌
    - 큰 weight에 punish

L1 regularization

$$E(\mathbf{p}, \mathbf{y}) = \frac{1}{2}(\mathbf{p} - \mathbf{y})^2 + \boxed{\lambda \sum |W_i|}$$

L2 regularization

$$E(\mathbf{p}, \mathbf{y}) = \frac{1}{2}(\mathbf{p} - \mathbf{y})^2 + \boxed{\lambda \sum W_i^2}$$

### CNNs

- Mainly used for processing images
- Perfirn better than MLP
- Less parameters than dense layers

#### Convolution

- Kernel - grid of weights
- Kernel is “applied” to the image
- Traditionally used in image processing

Image

| | | | | | |
|---|---|---|---|---|---|
| 5 | 2 | 3 | 1 | 2 | 4 |
| 2 | 4 | 1 | 0 | 3 | 1 |
| 5 | 1 | 0 | 2 | 8 | 3 |
| 0 | 2 | 1 | 5 | 2 | 4 |
| 2 | 7 | 0 | 0 | 2 | 1 |
| 1 | 3 | 2 | 8 | 7 | 0 |

Kernel

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 1 | 0 |
| 1 | 0 | -1 |

Output

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | ? | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

$$\sum_{i=1}^{P} image_i \cdot K_i = 5 \cdot 1 + 2 \cdot 0 + \ldots + 0 \cdot -1 = \boxed{18}$$

zero padding을 추가하면 엣지도 값을 구할 수 있음

Image

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 5 | 2 | 3 | 1 | 2 | 4 | 0 |
| 0 | 2 | 4 | 1 | 0 | 3 | 1 | 0 |
| 0 | 5 | 1 | 0 | 2 | 8 | 3 | 0 |
| 0 | 0 | 2 | 1 | 5 | 2 | 4 | 0 |
| 0 | 2 | 7 | 0 | 0 | 2 | 1 | 0 |
| 0 | 1 | 3 | 2 | 8 | 7 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Feature detectors
- Kernels are learned

Oblique line detector

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Vertical line detector

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 0 |

**Num of kernels**

- A conv layer has multiple kernels
- Each kernel outputs a single 2D array
- Output from a layer has as many 2d arrays as Num kernels

**Pooling**

- Downsample the image
- Overlaying gird on image
- Max/average pooling
- No parameters


Max pooling

cnn_genre_classifier.py

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATA_PATH = "../13/data_10.json"


def load_data(data_path):
    """Loads training dataset from json file.
        :param data_path (str): Path to json file containing data
        :return X (ndarray): Inputs
        :return y (ndarray): Targets
    """

    with open(data_path, "r") as fp:
        data = json.load(fp)

    X = np.array(data["mfcc"])
    y = np.array(data["labels"])
    return X, y


def plot_history(history):
    """Plots accuracy/loss for training/validation set as a function of the epochs
        :param history: Training history of model
        :return:
    """

    fig, axs = plt.subplots(2)

    # create accuracy sublpot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")

    # create error sublpot
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")

    plt.show()


def prepare_datasets(test_size, validation_size):
    """Loads data and splits it into train, validation and test sets.
    :param test_size (float): Value in [0, 1] indicating percentage of data set to allocate to test split
    :param validation_size (float): Value in [0, 1] indicating percentage of train set to allocate to validation split
    :return X_train (ndarray): Input training set
    :return X_validation (ndarray): Input validation set
    :return X_test (ndarray): Input test set
    :return y_train (ndarray): Target training set
    :return y_validation (ndarray): Target validation set
    :return y_test (ndarray): Target test set
    """

    # load data
    X, y = load_data(DATA_PATH)

    # create train, validation and test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validation_size)

    # add an axis to input sets
    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]

    return X_train, X_validation, X_test, y_train, y_validation, y_test


def build_model(input_shape):
    """Generates CNN model
    :param input_shape (tuple): Shape of input set
```

```
    :return model: CNN model
    """

    # build network topology
    model = keras.Sequential()

    # 1st conv layer
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # 2nd conv layer
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # 3rd conv layer
    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    # flatten output and feed it into dense layer
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.3))

    # output layer
    model.add(keras.layers.Dense(10, activation='softmax'))

    return model


def predict(model, X, y):
    """Predict a single sample using the trained model
    :param model: Trained classifier
    :param X: Input data
    :param y (int): Target
    """

    # add a dimension to input data for sample - model.predict() expects a 4d array in this case
    X = X[np.newaxis, ...] # array shape (1, 130, 13, 1)

    # perform prediction
    prediction = model.predict(X)

    # get index with max value
    predicted_index = np.argmax(prediction, axis=1)

    print("Target: {}, Predicted label: {}".format(y, predicted_index))


if __name__ == "__main__":

    # get train, validation, test splits
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.2)

    # create network
    input_shape = (X_train.shape[1], X_train.shape[2], 1)
    model = build_model(input_shape)

    # compile model
    optimiser = keras.optimizers.Adam(learning_rate=0.0001)
    model.compile(optimizer=optimiser,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.summary()

    # train model
    history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32, epochs=30)

    # plot accuracy/error for training and validation
    plot_history(history)

    # evaluate model on test set
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print('\nTest accuracy:', test_acc)

    # pick a sample to predict from the test set
    X_to_predict = X_test[100]
    y_to_predict = y_test[100]
```

```
# predict sample
predict(model, X_to_predict, y_to_predict)
```