## Class 2

# Power System Event Classification with Convolutional Neural Network Using PMU Data – Model Refinement & Assessment

## Outline

- Saving and loading models
- Metrics to assess performance
- Hyper-parameter tuning
- Loss curve and prevention of overfitting
- Assignment walk-through

## Outline

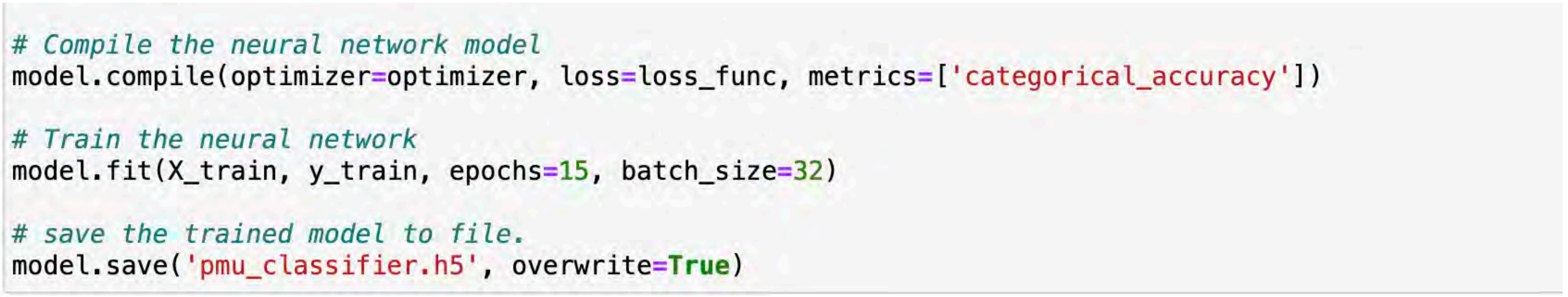
- Saving and loading models
- Metrics to assess performance
- Hyper-parameter tuning
- Loss curve and prevention of overfitting
- Assignment walk-through

# Saving and Loading Models

- Motivation
  - Machine learning models can take a very long time to train.
  - By saving and loading models, we can continue the unfinished training on another machine.
  - We can train on the powerful machine and deploy it in multiple systems when needed.
- Saving models requires the installation of the h5py library. It is usually installed as a dependency with TensorFlow. If it is not installed, you can install it in the Command Prompt with the command below.
  - sudo pip install h5py

# Save a Trained Model

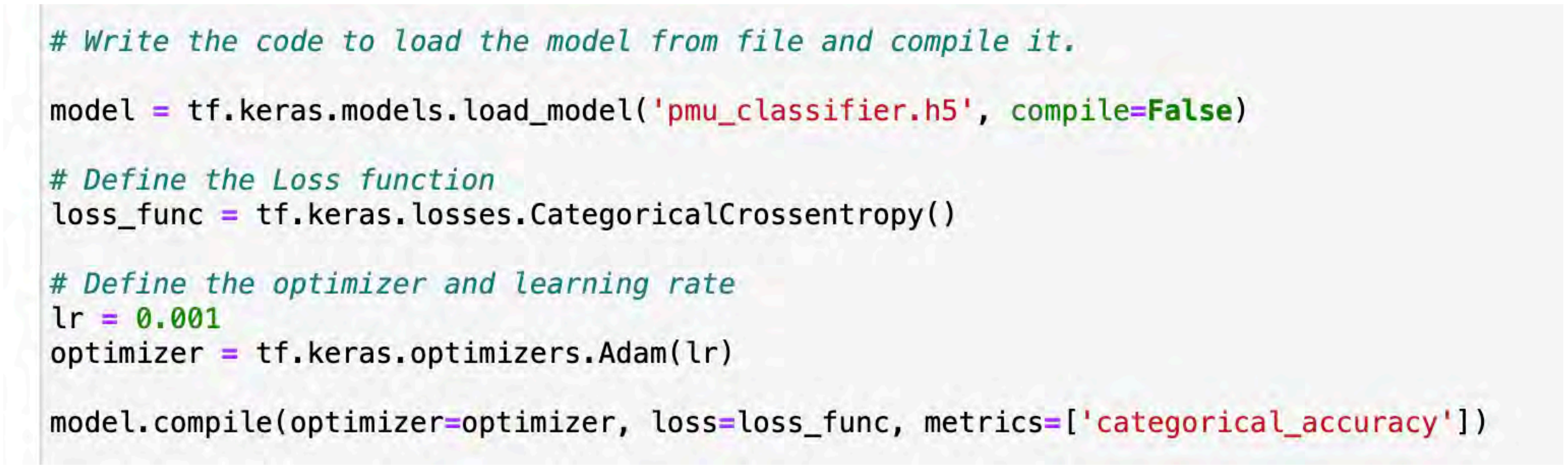
- To save a trained model, simply use the ".save(directory)" command in Tensorflow.
  - The method saves both the architecture configuration and weights of the model.
  - Out-of-the-box features provided by Tensorflow.
- Below is an incomplete toy example.
  - Here, we trained a CNN model named "model".
  - We save the trained model to the "pmu_classifier.h5" file.



```
# Compile the neural network model
model.compile(optimizer=optimizer, loss=loss_func, metrics=['categorical_accuracy'])

# Train the neural network
model.fit(X_train, y_train, epochs=15, batch_size=32)

# save the trained model to file.
model.save('pmu_classifier.h5', overwrite=True)
```

# Load a Saved Model

- We can load a saved model using the function “load_model”
- Below is an example.
  - We load the previously saved model from the file.
  - We define the loss function and optimizer, and compile the model for making predictions in the future, as well as evaluation or fine-tuning.



```
# Write the code to load the model from file and compile it.

model = tf.keras.models.load_model('pmu_classifier.h5', compile=False)

# Define the Loss function
loss_func = tf.keras.losses.CategoricalCrossentropy()

# Define the optimizer and learning rate
lr = 0.001
optimizer = tf.keras.optimizers.Adam(lr)

model.compile(optimizer=optimizer, loss=loss_func, metrics=['categorical_accuracy'])
```

## Outline

- Saving and loading models
- Metrics to assess performance
- Hyper-parameter tuning
- Loss curve and prevention of overfitting
- Assignment walk-through

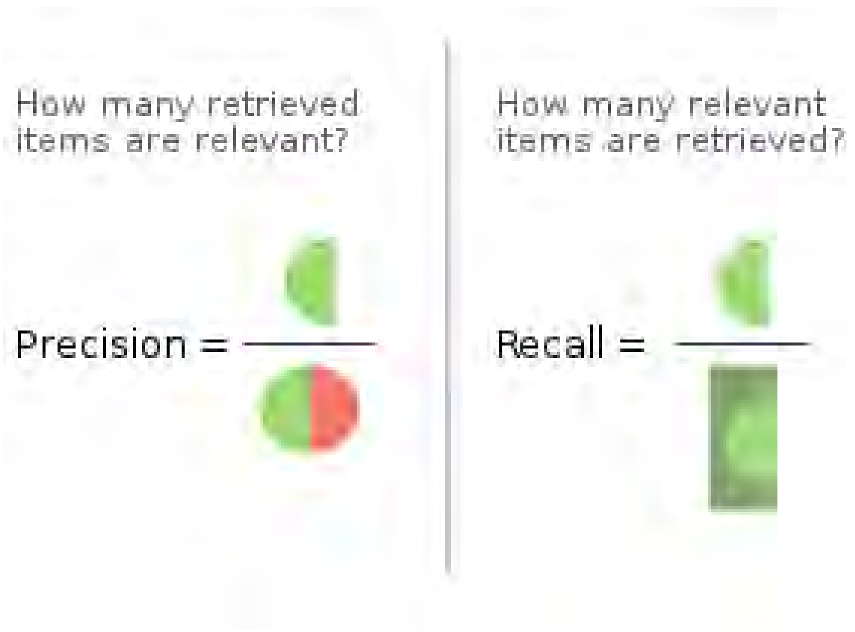
# Metrics Assess Performance

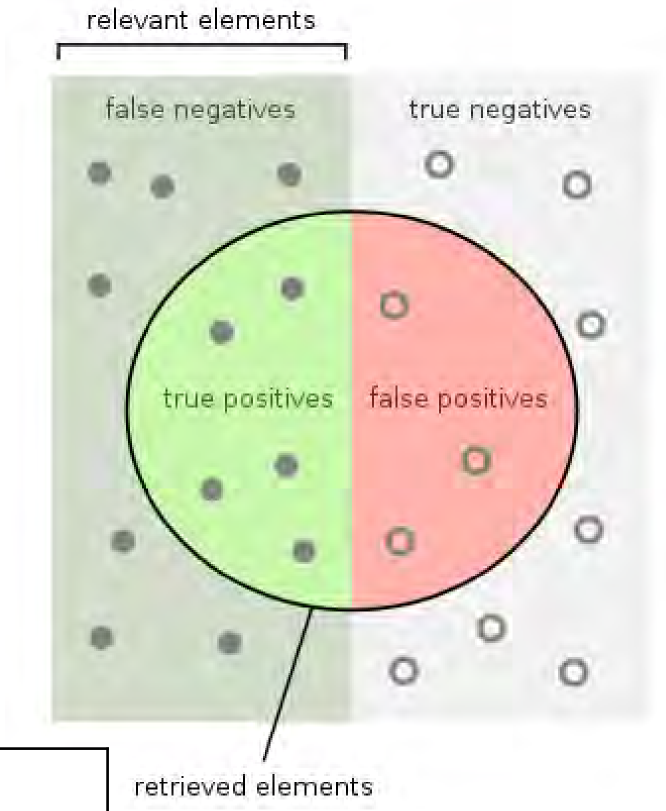
- There are four commonly used metrics to assess the performance of the classification model in this course.
  - Accuracy
  - Precision
  - Recall
  - F-1 Score
- Confusion Matrix (Performance Matrix, Error Table)

| | | **Predicted** | |
|---|---|---|---|
| | | **Negative** | **Positive** |
| **Actual** | **Negative** | True Negative | False Positive |
| | **Positive** | False Positive | True Negative |

## Calculate Metrics Assess Performance

- Accuracy: $\frac{TN+TP}{TN+TP+FN+FP}$
- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- F1-Score: $\frac{2*Precision*Recall}{Precision+Recall}$

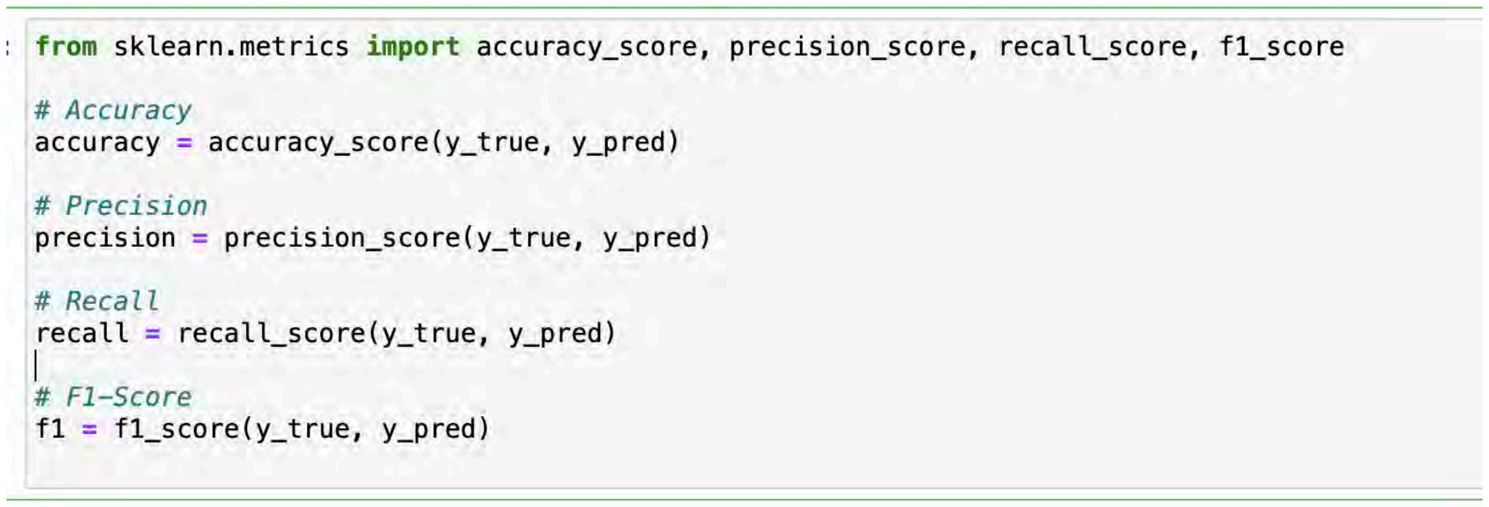

How many retrieved items are relevant?
Precision =
How many relevant items are retrieved?
Recall =



| | | **Negative** | **Positive** |
|---|---|---|---|
| **Actual** | **Negative** | True Negative (TN) | False Positive (FP) |
| | **Positive** | False Negative (FN) | True Negative (TN) |

## Python code example of metrics

- Scikit-learn is a Python library that provides a wide range of useful tools for machine learning.
- Here are some example codes for calculating evaluation metrics in Python.
- More useful and in-depth resources can be found here:
  - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
  - https://scikit-learn.org/stable/auto_examples/index.html



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Accuracy
accuracy = accuracy_score(y_true, y_pred)

# Precision
precision = precision_score(y_true, y_pred)

# Recall
recall = recall_score(y_true, y_pred)

# F1-Score
f1 = f1_score(y_true, y_pred)
```

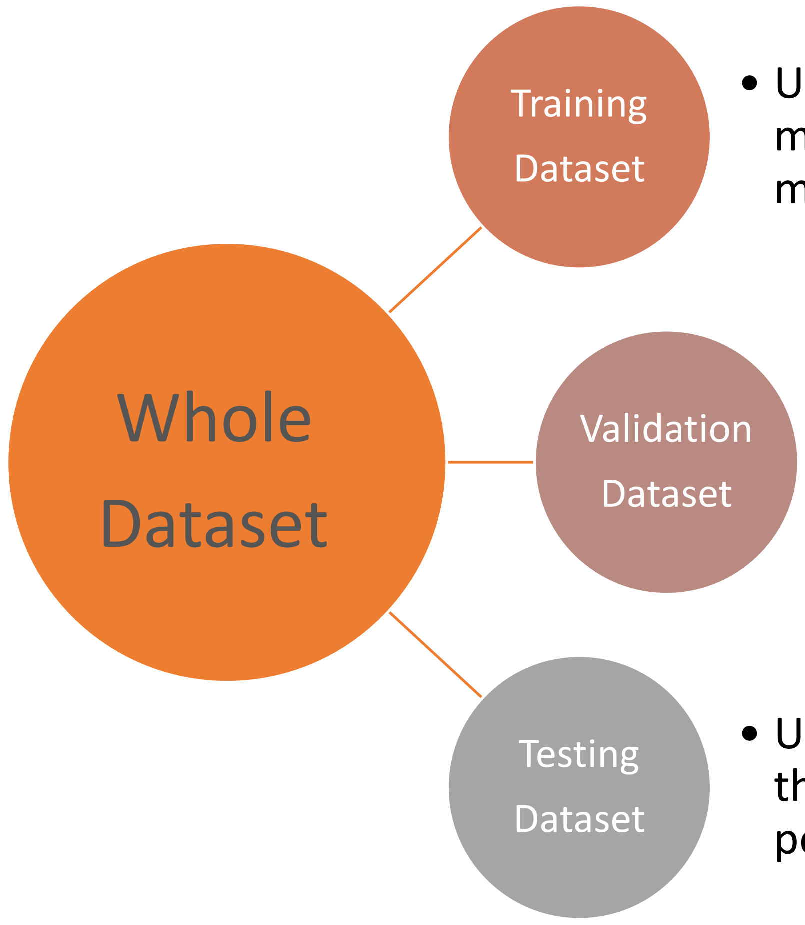
# Outline

- Saving and loading models
- Metrics to assess performance
- Hyper-parameter tuning
- Loss curve and prevention of overfitting
- Assignment walk-through

## Hyper-Parameter Tuning

- Hyper-parameters are the settings and parameters that control the configuration of machine learning models.
- They cannot be trained in the model training procedure.
- They influence the performance of the models.
- Examples of hyper-parameters for a CNN model:
    - Batch size.
    - Learning rate.
    - Number of the training epoch.
    - The number of hidden Convolutional layers.
    - Filter Size and Stride of each Convolutional layer.

## Dataset Split



- Used to train machine learning models.
- Used for hyperparameter tuning and early stopping.
- Used to evaluate the prediction performance.

## Hyper-Parameter Tuning with Validation Set

- We split the dataset into three parts: training set, validation set, and testing set.
- For each hyper-parameter setup, we train the model with the training set and evaluate its performance with the validation set.
- The setup with the best performance (lowest loss, etc.) in the validation set is selected as the optimal hyper-parameter setup.
- Do not use testing dataset during Training and Hyper-parameter tuning!

## Grid Search

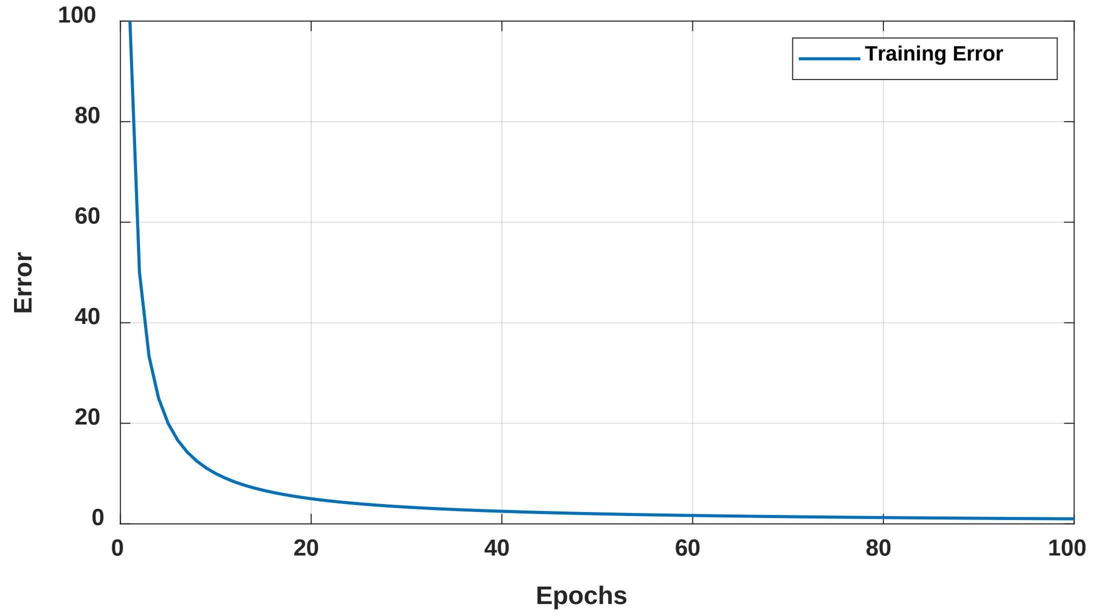
- Each hyper-parameter has a list of possible values.
- Evaluate the performance of all the possible combinations (“grid”) of hyper-parameter values.
- Example 1: suppose batch size = [5, 10, 20], learning rate = [0.01, 0.001], then the grid has $3 \times 2 = 6$ possible combinations.
- Example 2: if there is one additional hyper-parameter for the number of convolutional layers =[3, 5, 6], then the grid has $3 \times 2 \times 3 = 18$ combinations.

# Outline

- Saving and loading models
- Metrics to assess performance
- Hyper-parameter tuning
- Loss curve and prevention of overfitting
- Assignment walk-through

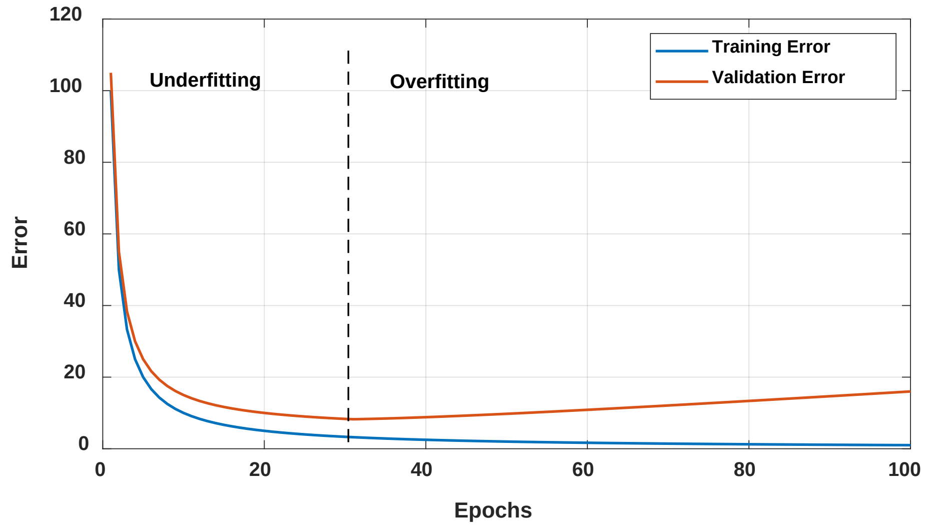
# Loss Curve

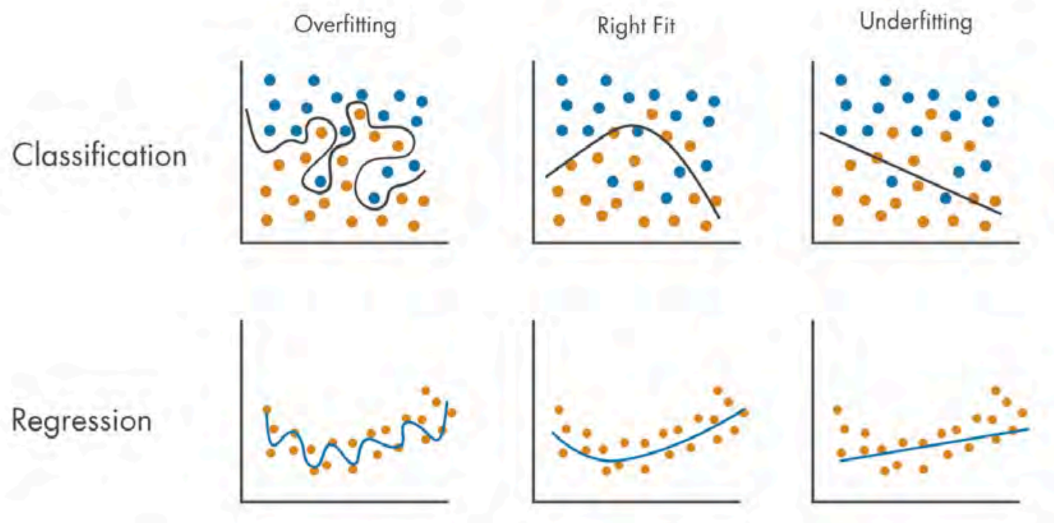
- The training process of a machine learning model is to minimize a loss of function value.
- Example:
  - CrossEntropy loss of the model’s output with the target values in the training dataset.
  - As the training process goes on, the loss value generally decreases.
  - Observing the loss can help identify **overfitting**, which will be explained on the next slide.



# Overfitting

- We need to make sure the model not only works well in the training dataset but also works well in the dataset that is not seen in the training.
- If the model learns the training dataset so well that it learns the dataset's unnecessary details and noises, the model may not work well in the unseen dataset. This is called "Overfitting".



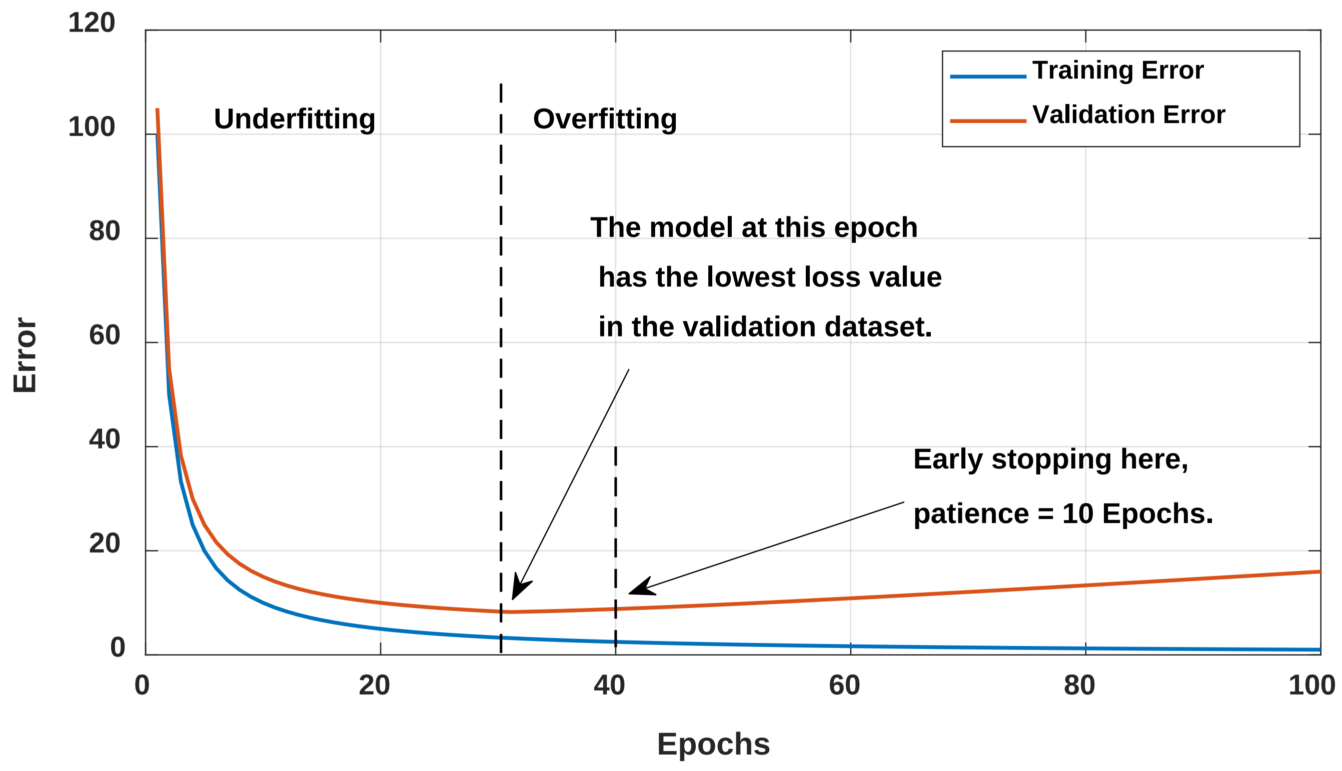


Source: https://www.mathworks.com/discovery/overfitting.html

# Prevention of Overfitting

- We can use the performance on the validation dataset to evaluate how well the trained model works on unseen datasets.
- During the training process, monitor the loss value in the validation dataset. If the loss value starts to increase, we know overfitting is happening.
- This is called “Early Stopping”.
    - Set a value p called patience. This is the number of iterations or epochs that will be completed before we stop the training.
    - Train the model with the training dataset. Monitor the loss value on the validation dataset.
    - If the loss value on the validation dataset no longer improved for continuous *p* iterations/epochs, then we stop the training.

# Early Stopping Example

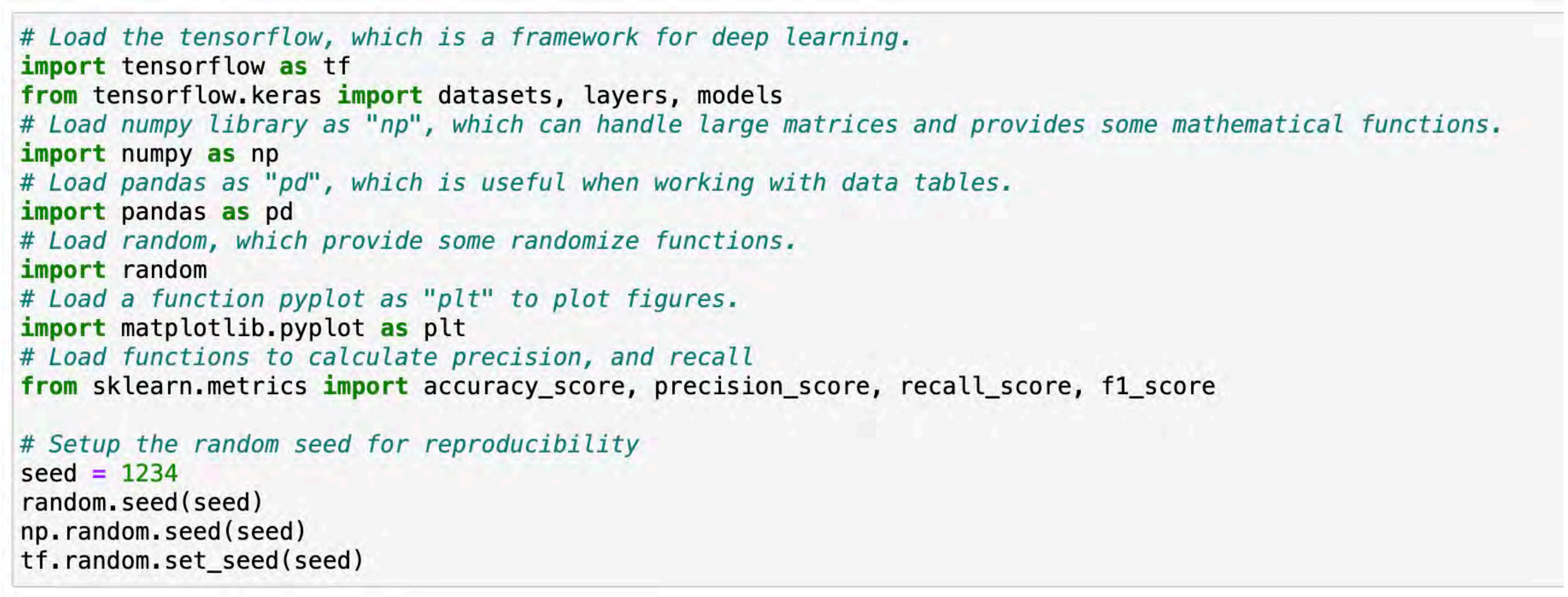


## Outline

- Saving and loading models
- Metrics to assess performance
- Hyper-parameter tuning
- Loss curve and prevention of overfitting
- Assignment walk-through

## Load Needed Libraries/Packages

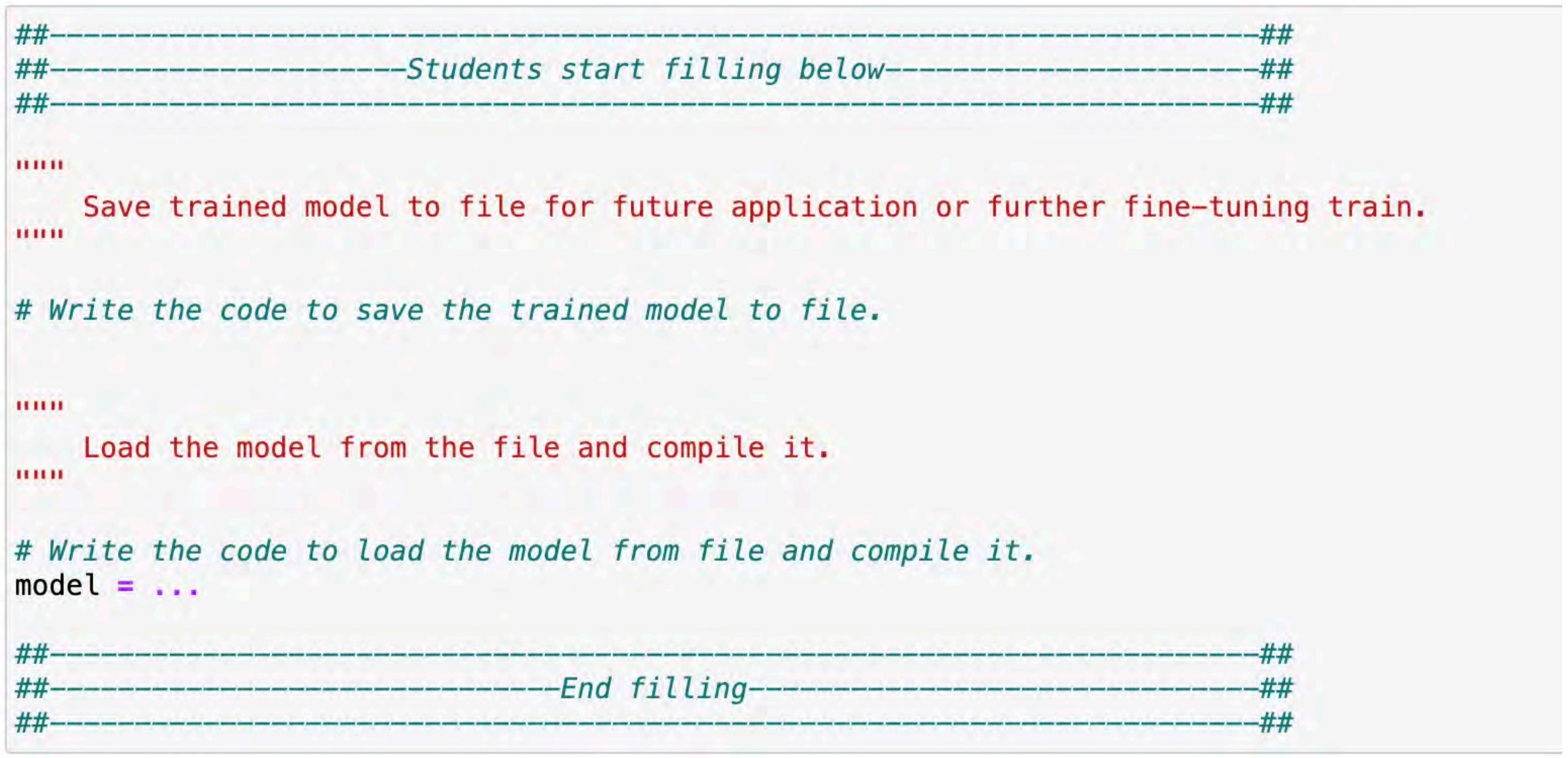
- To start, we need to load the libraries and packages that will used in the code.
- Similar to class 1, but we need new packages for evaluation metrics and early stopping.



```
# Load the tensorflow, which is a framework for deep learning.
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
# Load numpy library as "np", which can handle large matrices and provides some mathematical functions.
import numpy as np
# Load pandas as "pd", which is useful when working with data tables.
import pandas as pd
# Load random, which provide some randomize functions.
import random
# Load a function pyplot as "plt" to plot figures.
import matplotlib.pyplot as plt
# Load functions to calculate precision, and recall
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Setup the random seed for reproducibility
seed = 1234
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)
```

## Saving and Loading Model

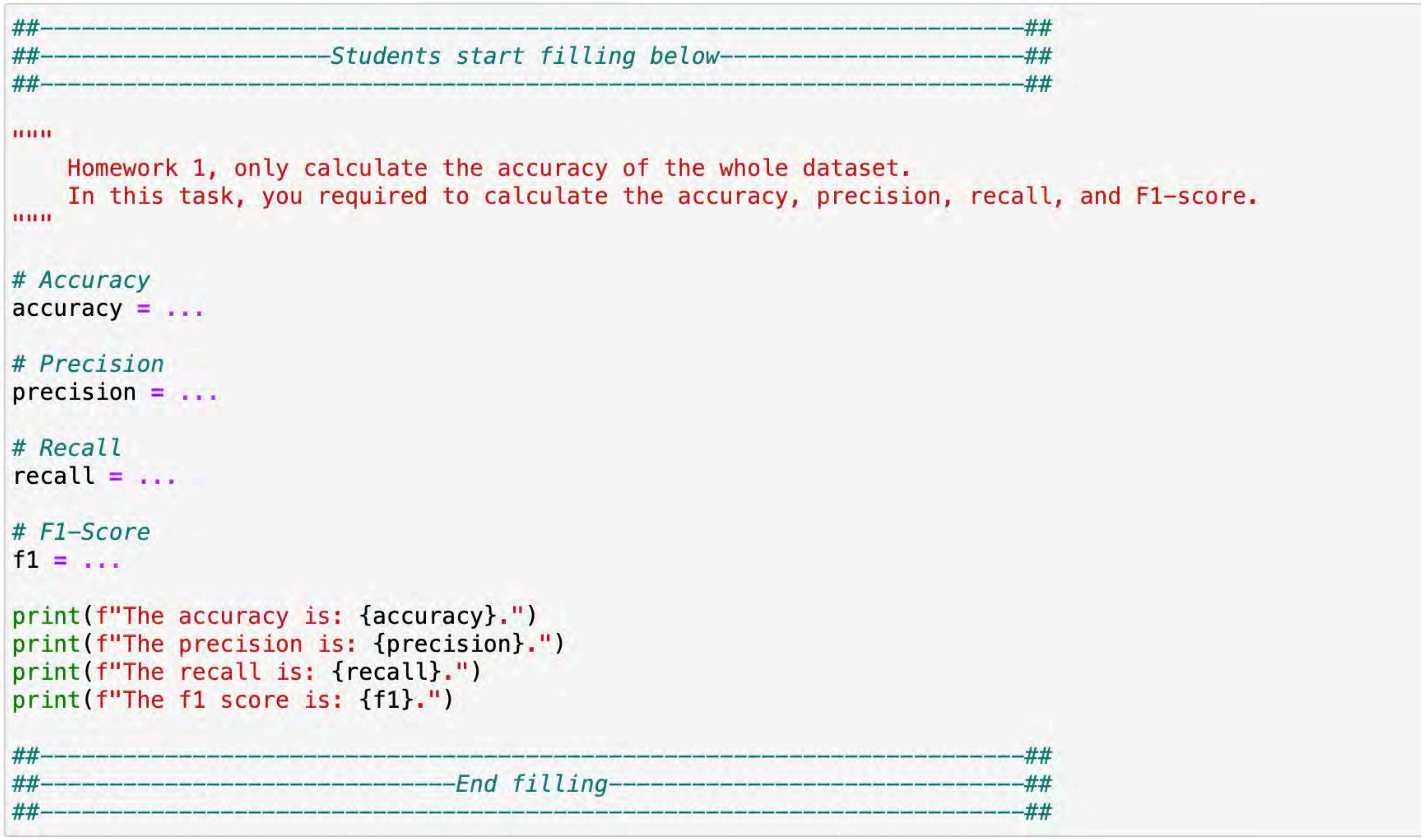
- Save the trained model to file for future application or further fine-tuning.
- Load the model from the file and compile it.



```
##-----------------------------------------------------------------------##
##---------------------Students start filling below----------------------##
##-----------------------------------------------------------------------##

"""
    Save trained model to file for future application or further fine-tuning train.
"""

# Write the code to save the trained model to file.


"""
    Load the model from the file and compile it.
"""

# Write the code to load the model from file and compile it.
model = ...

##-----------------------------------------------------------------------##
##------------------------------End filling------------------------------##
##-----------------------------------------------------------------------##
```

# Performance Assessment using Metrics

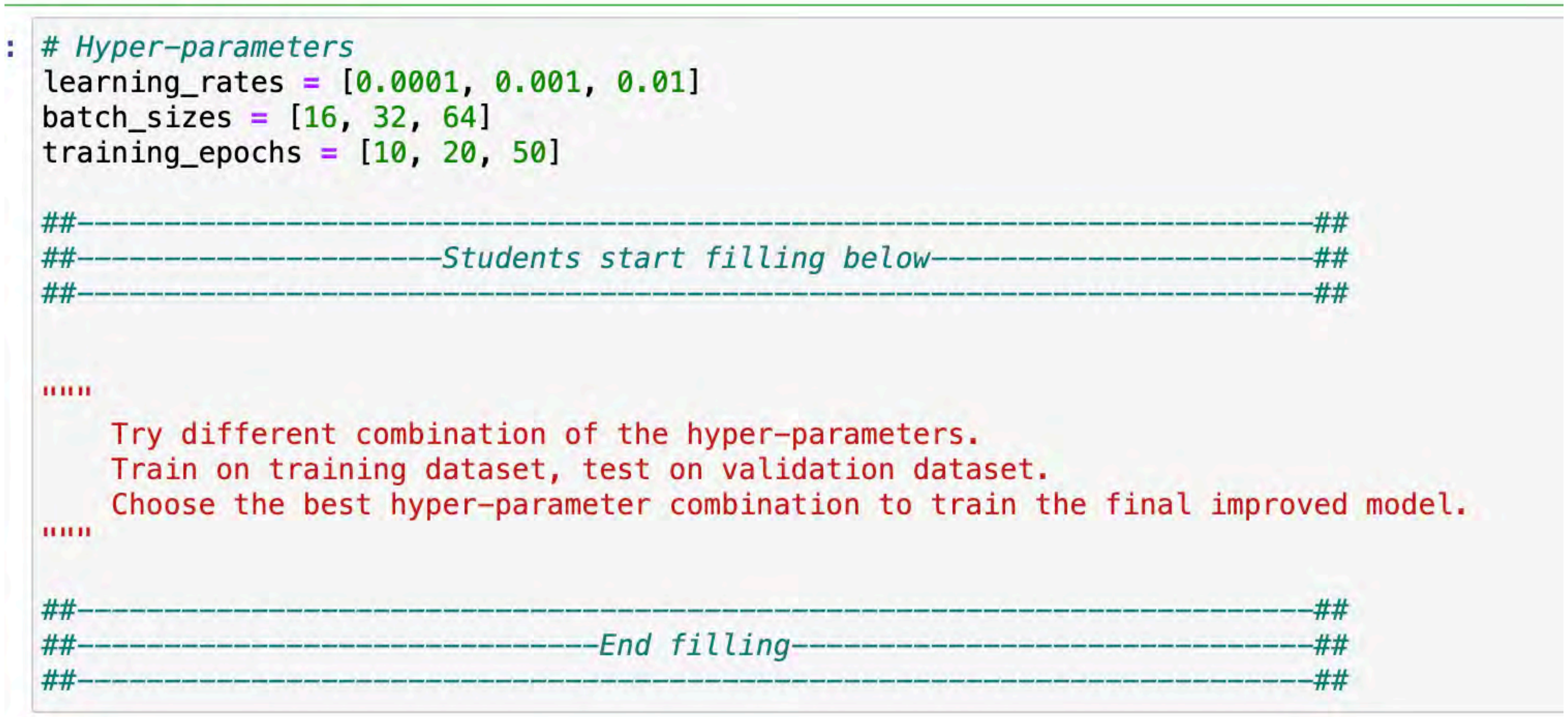
- In this task, you are required to calculate the precision, recall, and F1-score.



```
##------------------------------------------------------------------------##
##--------------------Students start filling below----------------------##
##------------------------------------------------------------------------##

"""
    Homework 1, only calculate the accuracy of the whole dataset.
    In this task, you required to calculate the accuracy, precision, recall, and F1-score.
"""

# Accuracy
accuracy = ...

# Precision
precision = ...

# Recall
recall = ...

# F1-Score
f1 = ...

print(f"The accuracy is: {accuracy}.")
print(f"The precision is: {precision}.")
print(f"The recall is: {recall}.")
print(f"The f1 score is: {f1}.")

##------------------------------------------------------------------------##
##----------------------------End filling-------------------------------##
##------------------------------------------------------------------------##
```

## Grid Search for Hyper-Parameter Tuning

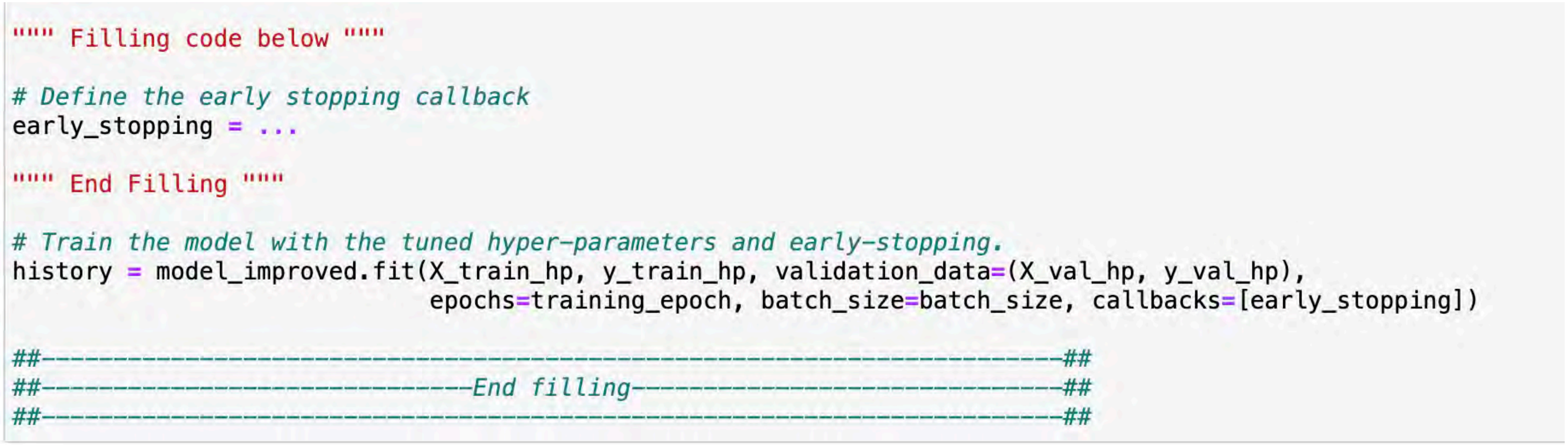
- You are required to build your code to find the best hyperparameter settings that can achieve the best performance (lowest loss) on the validation dataset.
- In your code, you need to do a grid search of the learning rate, batch size, training epochs, and any other hyper-parameters you like.



```
# Hyper-parameters
learning_rates = [0.0001, 0.001, 0.01]
batch_sizes = [16, 32, 64]
training_epochs = [10, 20, 50]

##-----------------------------------------------------------------------##
##--------------------Students start filling below---------------------##
##-----------------------------------------------------------------------##


"""
    Try different combination of the hyper-parameters.
    Train on training dataset, test on validation dataset.
    Choose the best hyper-parameter combination to train the final improved model.
"""


##-----------------------------------------------------------------------##
##-----------------------------End filling-----------------------------##
##-----------------------------------------------------------------------##
```

## Set Up the Early Stopping

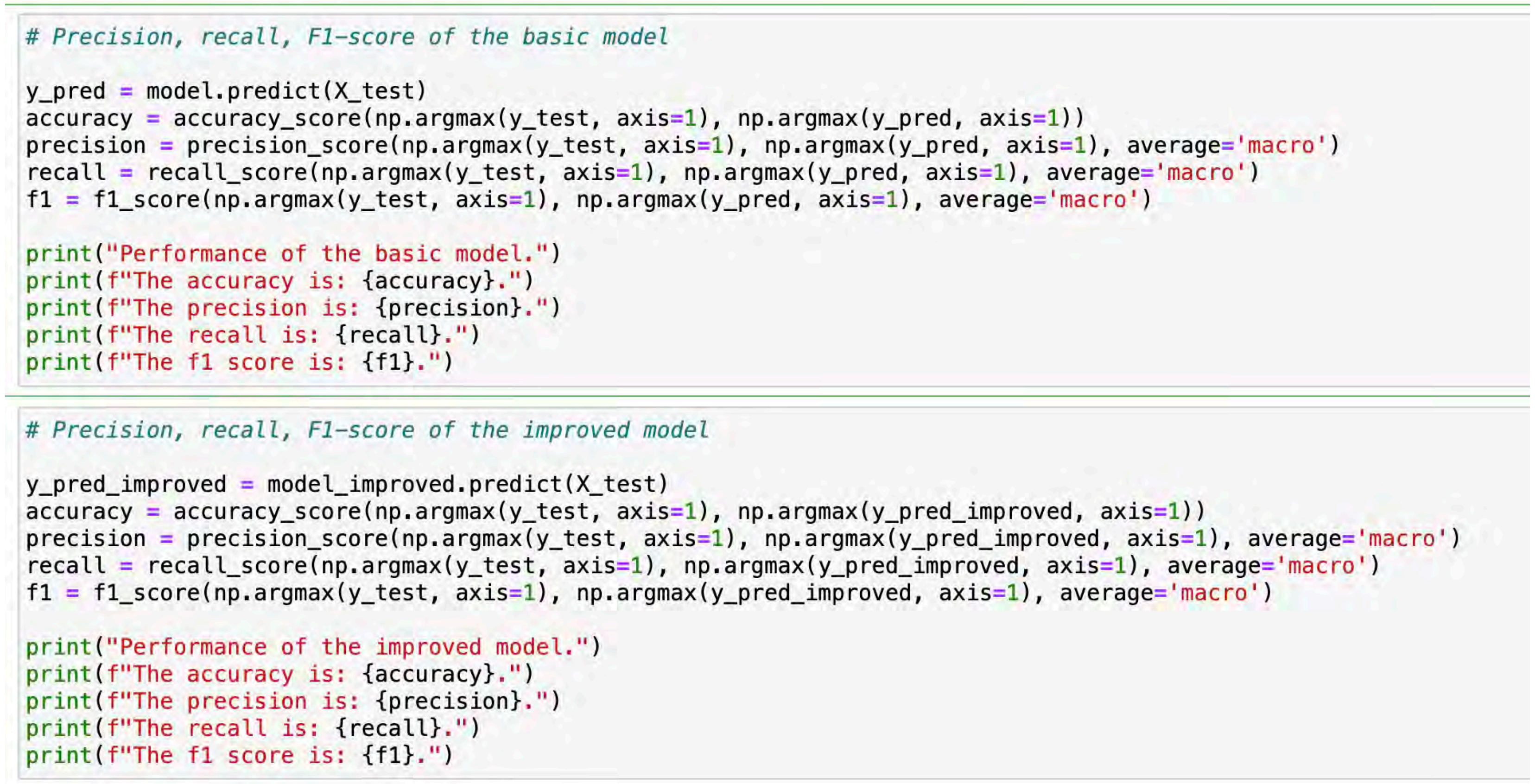
- This task needs to setup the early stopping to prevent overfitting.
- Useful resource: https://keras.io/api/callbacks/early_stopping/



```
""" Filling code below """

# Define the early stopping callback
early_stopping = ...

""" End Filling """

# Train the model with the tuned hyper-parameters and early-stopping.
history = model_improved.fit(X_train_hp, y_train_hp, validation_data=(X_val_hp, y_val_hp),
                             epochs=training_epoch, batch_size=batch_size, callbacks=[early_stopping])

##---------------------------------------------------------------------##
##-----------------------------End filling-----------------------------##
##---------------------------------------------------------------------##
```

## Compare With Basic Model

- After hyper-parameter tuning, and early stopping.
- Compare with the performance of the basic model and report the result.



```
# Precision, recall, F1-score of the basic model

y_pred = model.predict(X_test)
accuracy = accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1))
precision = precision_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1), average='macro')
recall = recall_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1), average='macro')
f1 = f1_score(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1), average='macro')

print("Performance of the basic model.")
print(f"The accuracy is: {accuracy}.")
print(f"The precision is: {precision}.")
print(f"The recall is: {recall}.")
print(f"The f1 score is: {f1}.")
```

```
# Precision, recall, F1-score of the improved model

y_pred_improved = model_improved.predict(X_test)
accuracy = accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_pred_improved, axis=1))
precision = precision_score(np.argmax(y_test, axis=1), np.argmax(y_pred_improved, axis=1), average='macro')
recall = recall_score(np.argmax(y_test, axis=1), np.argmax(y_pred_improved, axis=1), average='macro')
f1 = f1_score(np.argmax(y_test, axis=1), np.argmax(y_pred_improved, axis=1), average='macro')

print("Performance of the improved model.")
print(f"The accuracy is: {accuracy}.")
print(f"The precision is: {precision}.")
print(f"The recall is: {recall}.")
print(f"The f1 score is: {f1}.")
```