```python
from numpy import loadtxt #handle/load dataset
from keras.models import Sequential #Empty working area
from keras.layers import Dense #Dense layer
from keras.layers import BatchNormalization

dataset = loadtxt('dataset.csv', delimiter=',')
x = dataset[:,0:8]
y = dataset[:,8]
print(x)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)

model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1, activation='sigmoid'))
```

```
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x, y, epochs=5, batch_size=10)

_, accuracy = model.evaluate(x, y)
print('Accuracy: %.2f' % (accuracy*100))

from sklearn.metrics import accuracy_score
import numpy as np
y_pred = model.predict(x)
pred = list()
for i in range(len(y_pred)):
    pred.append(np.argmax(y_pred[i]))
test=y
a = accuracy_score(pred,test)
print('Accuracy is:', a*100)


predictions = model.predict_classes(x)
```

```
test=y
a = accuracy_score(pred,test)
print('Accuracy is:', a*100)


predictions = model.predict_classes(x)
for i in range(5,10):
    print('Predicted Class: %d (Original Class: %d)' % (predictions[i], y[i]))

model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")

```

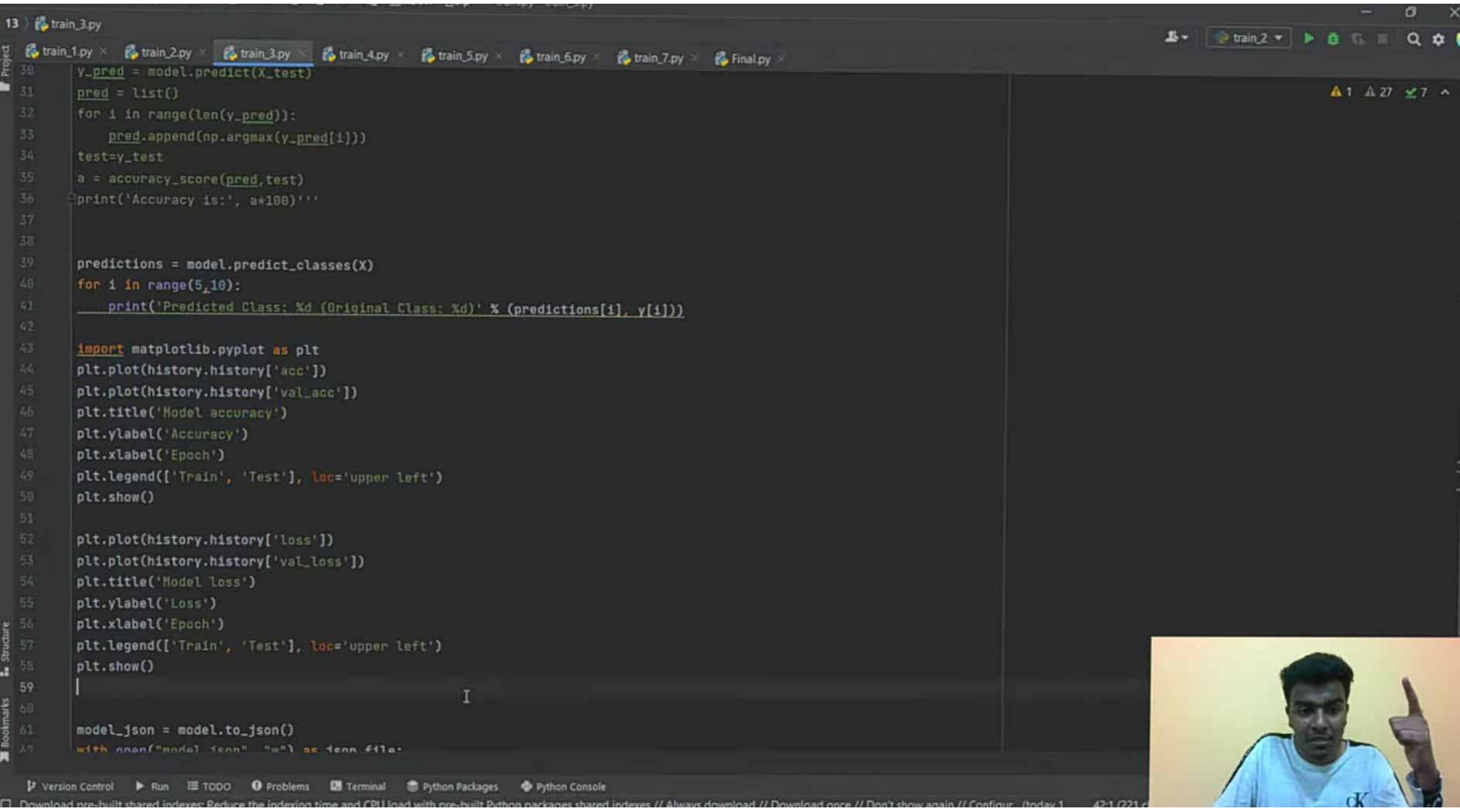
File Edit View Navigate Code Refactor Run Tools VCS Window Help train.py - train_3.py

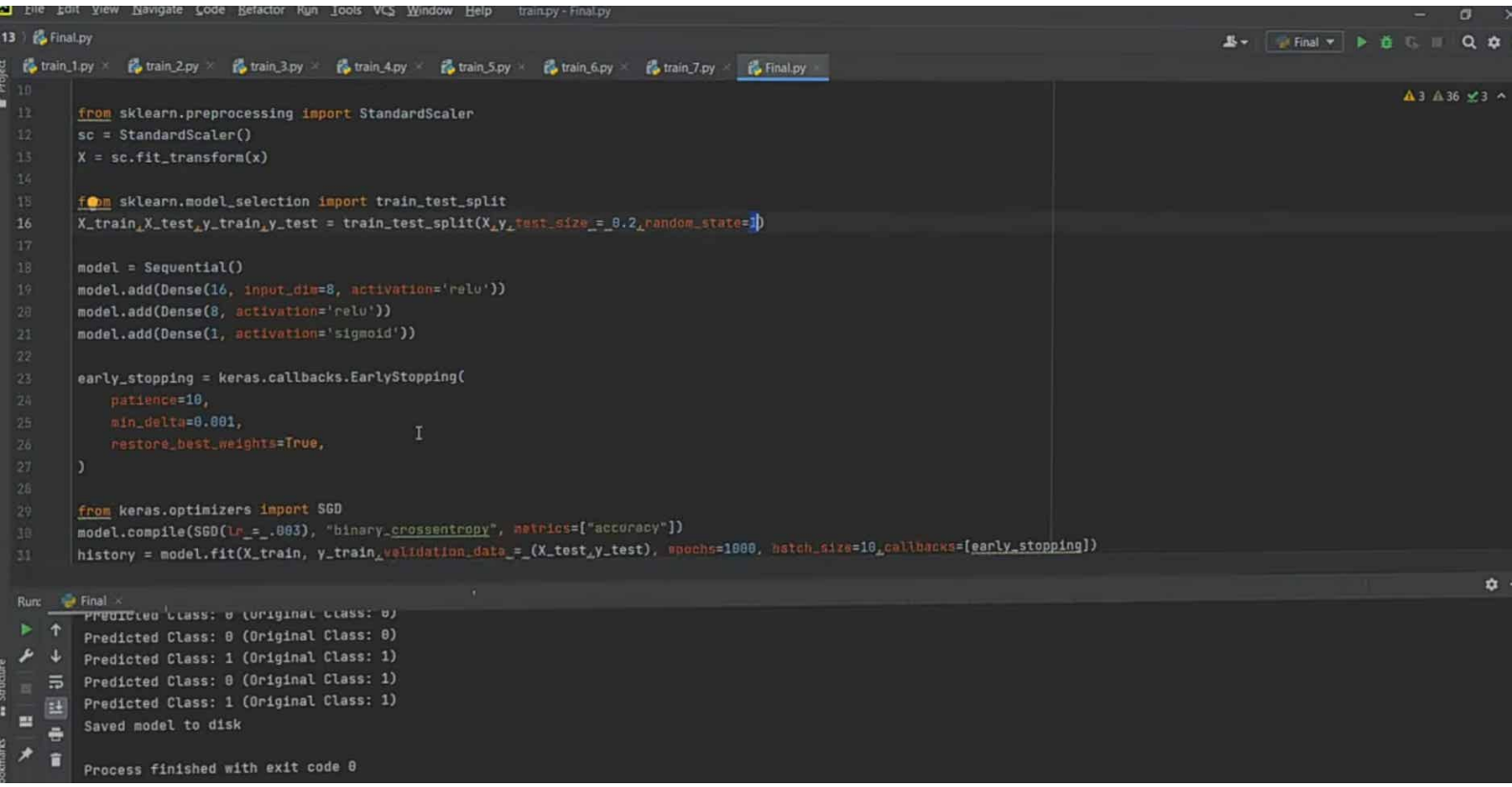
train_3.py

train_1.py | train_2.py | train_3.py | train_4.py | train_5.py | train_6.py | train_7.py | Final.py

```python
from numpy import loadtxt #handle/load dataset
from keras.models import Sequential #Empty working area
from keras.layers import Dense #Dense layer

dataset = loadtxt('dataset.csv', delimiter=',')
x = dataset[:,0:8]
y = dataset[:,8]

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(x)

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=1)

model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, y_train,validation_data=(X_test,y_test), epochs=100, batch_size=10)

_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))


'''from sklearn.metrics import accuracy_score
import numpy as np
y_pred = model.predict(X_test)
pred = list()
for i in range(len(y_pred)):
```

Version Control | Run | TODO | Problems | Terminal | Python Packages | Python Console


train_3.py
train_1.py
train_2.py
train_3.py
train_4.py
train_5.py
train_6.py
train_7.py
Final.py
train_2
y_pred = model.predict(X_test)
pred = list()
for i in range(len(y_pred)):
pred.append(np.argmax(y_pred[i]))
test=y_test
a = accuracy_score(pred,test)
print('Accuracy is:', a*100)'''
predictions = model.predict_classes(X)
for i in range(5,10):
print('Predicted Class: %d (Original Class: %d)' % (predictions[i], y[i]))
import matplotlib.pyplot as plt
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
model_json = model.to_json()
Version Control
Run
TODO
Problems
Terminal
Python Packages
Python Console


File Edit View Navigate Code Refactor Run Tools VCS Window Help train.py - Final.py
Final.py
train_1.py train_2.py train_3.py train_4.py train_5.py train_6.py train_7.py Final.py
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(x)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=1)
model = Sequential()
model.add(Dense(16, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
early_stopping = keras.callbacks.EarlyStopping(
patience=10,
min_delta=0.001,
restore_best_weights=True,
)
from keras.optimizers import SGD
model.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
history = model.fit(X_train, y_train, validation_data = (X_test,y_test), epochs=1000, batch_size=10, callbacks=[early_stopping])
Run: Final
Predicted Class: 0 (Original Class: 0)
Predicted Class: 0 (Original Class: 0)
Predicted Class: 1 (Original Class: 1)
Predicted Class: 0 (Original Class: 1)
Predicted Class: 1 (Original Class: 1)
Saved model to disk
Process finished with exit code 0

Deep Learning Terminology - 1

Handling overfitting

- **Simplifying The Model:** Reduce the network's capacity by removing layers or reducing the number of elements in the hidden layers
- Apply regularization, which comes down to adding a cost to the loss function for large weights
  - ✓ L1 Regularization
  - ✓ L2 Regularization
- Use Dropout layers, which will randomly remove certain features by setting them to zero
- Early Stopping
- Use Data Augmentation

L1 Regularization	L2 Regularization
1. L1 penalizes sum of absolute values of weights.	1. L2 penalizes sum of square values of weights.
2. L1 generates model that is simple and interpretable.	2. L2 regularization is able to learn complex data patterns.
3. L1 is robust to outliers.	3. L2 is not robust to outliers.

Deep Learning Terminology - 2

Outliers

- Values distant from most other values. In machine learning, any of the following are outliers:
- Weights with high absolute values.
- Predicted values relatively far away from the actual values.
- Input data whose values are more than roughly 3 standard deviations from the mean.

Deep Learning Terminology - 3

Oversampling

- Reusing the examples of a minority class in a class-imbalanced dataset in order to create a more balanced training set.
- We need to be careful about over overfitting when oversampling.
- Contrast with undersampling.