Hands_on_Activity_8_1

Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
Summer	AY 2024 - 2025

Hands-on Activity 8.1	**Saving Models**
Name	Calvadores, Kelly Joseph
Section	CPE32S1
Date Performed:	July 01, 2024
Date Submitted:	July 05, 2024
Instructor:	Engr. Roman M. Richard

Choose any dataset applicable to either a classification problem or a regression problem.¶

Explain your datasets and the problem being addressed.¶

- The problem that is currently being addressed is to build a model that can classify the different types of glass based on the given Dataset.

Show evidence that you can do the following:¶

In [18]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

```
from keras.utils import to_categorical
import os
```

Resource: https://archive.ics.uci.edu/dataset/42/glass+identification

In [19]:

```
ColumnNames = ['Id_number', 'RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba',
'Fe', 'Type_of_glass']
Data = pd.read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/glass.data', header=None)
Data.columns = ColumnNames
Data.to_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/glass_data_with_header.csv', index=False)
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id_number      214 non-null    int64
 1   RI             214 non-null    float64
 2   Na             214 non-null    float64
 3   Mg             214 non-null    float64
 4   Al             214 non-null    float64
 5   Si             214 non-null    float64
 6   K              214 non-null    float64
 7   Ca             214 non-null    float64
 8   Ba             214 non-null    float64
 9   Fe             214 non-null    float64
 10  Type_of_glass  214 non-null    int64
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

In [20]:

```
Data
```

Out[20]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1

3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [21]:

```
X= Data.iloc[:, :-1]
y = Data.iloc[:, -1]
```

In [22]:

```
SS = StandardScaler()
X = SS.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.001,
random_state=123)

LE = LabelEncoder()
y_train = LE.fit_transform(y_train)
y_test = LE.fit_transform(y_test)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [23]:

```
Model = Sequential()
Model.add(Dense(16, input_dim=10, activation='relu'))
Model.add(Dense(8, activation='relu'))
Model.add(Dense(6, activation='sigmoid'))
```

In [24]:

```
Model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
Model.fit(X_train, y_train, epochs=150, batch_size=1000, verbose = 0)
Result = Model.evaluate(X_train, y_train)
print("%s: %.2f%%" % (Model.metrics_names[1], Result[1]*100))
```

```
7/7 [==============================] - 0s 4ms/step - loss: 0.7395 - accuracy:
0.7934
accuracy: 79.34%
```

Observation: When the first run on this model, the result is below 35%, but when I did some scaling and splitting data, it rose to 79%, both accuracy and loss has little difference of 6%

Save a model and load the model in a JSON format¶

In [25]:

```
Model_json = Model.to_json()
with open("/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/Models/Model.json", "w") as json_file:
    json_file.write(Model_json)
    print("Saved model to disk")
```

```
Saved model to disk
```

In [26]:

```
Model.save_weights("/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/Models/Model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

In [28]:

```
from tensorflow.keras.models import Sequential, model_from_json
json_file = open('/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/Models/Model.json', 'r')
Loaded_model_json = json_file.read()
json_file.close()
Loaded_model = model_from_json(Loaded_model_json)
Loaded_model.load_weights("/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/Models/Model.h5")
print("Loaded model from disk")
```

```
Loaded model from disk
```

In [29]:

```
Loaded_model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
Result = Loaded_model.evaluate(X_train, y_train, verbose = 0)
print("%s: %.2f%%" % (Loaded_model.metrics_names[1], Result[1]*100))
```

```
accuracy: 79.34%
```

Observation: In this part of code, saving and loading model is an amazing thing when creating a project, when loading the model, same accuracy was made.

Save a model and load the model in a YAML format¶

In [30]:

```
from tensorflow.keras.models import Sequential, model_from_yaml
Model_yaml = Model.to_json()
with open("/content/drive/MyDrive/CPE 019 (Retake)/HOA 
8.1/Models/Model.yaml", "w") as yaml_file:
    yaml_file.write(Model_yaml)
Model.save_weights("/content/drive/MyDrive/CPE 019 (Retake)/HOA 
8.1/Models/Model_yaml.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

In [31]:

```
yaml_file = open('/content/drive/MyDrive/CPE 019 (Retake)/HOA 
8.1/Models/Model.yaml', 'r')
Loaded_model_yaml = yaml_file.read()
yaml_file.close()
Loaded_model = model_from_json(Loaded_model_yaml)
Loaded_model.load_weights("/content/drive/MyDrive/CPE 019 (Retake)/HOA 
8.1/Models/Model_yaml.h5")
print("Loaded model from disk")
Loaded_model.compile(loss='categorical_crossentropy', optimizer='adam', 
metrics=['accuracy'])
Result = Loaded_model.evaluate(X_train, y_train, verbose = 0)
print("%s: %.2f%%" % (Loaded_model.metrics_names[1], Result[1]*100))
```

```
Loaded model from disk
accuracy: 79.34%
```

Observation: The yaml file is almost same process as json file except .yaml is used, when is serch it, it was because yaml module is not installed in this google colab

Checkpoint Neural Network Model Improvements¶

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

tf.random.set_seed(42)
Data = pd.read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/glass_data_with_header.csv')
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id_number      214 non-null    int64
 1   RI             214 non-null    float64
 2   Na             214 non-null    float64
 3   Mg             214 non-null    float64
 4   Al             214 non-null    float64
 5   Si             214 non-null    float64
 6   K              214 non-null    float64
 7   Ca             214 non-null    float64
 8   Ba             214 non-null    float64
 9   Fe             214 non-null    float64
 10  Type_of_glass  214 non-null    int64
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

In [ ]:

```
X = Data.iloc[:, :-1]
y = Data.iloc[:, -1]

SS = StandardScaler()
X = SS.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=123)

LE = LabelEncoder()
y_train = LE.fit_transform(y_train)
y_test = LE.fit_transform(y_test)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

Model = Sequential()
Model.add(Dense(64, input_dim=10, activation='relu'))
Model.add(Dense(32, activation='relu'))
```

```
Model.add(Dense(6, activation='softmax'))

Model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [ ]:

```
Filepath = "/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-{epoch:02d}-{val_accuracy:.2f}.keras"
CheckPoint = ModelCheckpoint(filepath=Filepath, monitor='val_accuracy',
verbose=1, save_best_only=True, mode='max')
Callbacks_list = [CheckPoint]

Model.fit(X_train, y_train, validation_split=0.33, epochs=150,
batch_size=1000, callbacks=Callbacks_list, verbose=0)
```

```
Epoch 1: val_accuracy improved from -inf to 0.20833, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-01-0.21.keras

Epoch 2: val_accuracy improved from 0.20833 to 0.25000, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-02-0.25.keras

Epoch 3: val_accuracy improved from 0.25000 to 0.27083, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-03-0.27.keras

Epoch 4: val_accuracy improved from 0.27083 to 0.31250, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-04-0.31.keras

Epoch 5: val_accuracy improved from 0.31250 to 0.37500, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-05-0.38.keras

Epoch 6: val_accuracy improved from 0.37500 to 0.39583, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-06-0.40.keras

Epoch 7: val_accuracy improved from 0.39583 to 0.45833, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-07-0.46.keras
```

```
Epoch 8: val_accuracy improved from 0.45833 to 0.50000, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-08-0.50.keras

Epoch 9: val_accuracy did not improve from 0.50000

Epoch 10: val_accuracy did not improve from 0.50000

Epoch 11: val_accuracy improved from 0.50000 to 0.52083, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-11-0.52.keras

Epoch 12: val_accuracy did not improve from 0.52083

Epoch 13: val_accuracy did not improve from 0.52083

Epoch 14: val_accuracy did not improve from 0.52083

Epoch 15: val_accuracy improved from 0.52083 to 0.54167, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-15-0.54.keras

Epoch 16: val_accuracy improved from 0.54167 to 0.58333, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-16-0.58.keras

Epoch 17: val_accuracy improved from 0.58333 to 0.62500, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-17-0.62.keras

Epoch 18: val_accuracy improved from 0.62500 to 0.64583, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-18-0.65.keras

Epoch 19: val_accuracy improved from 0.64583 to 0.66667, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-19-0.67.keras

Epoch 20: val_accuracy improved from 0.66667 to 0.68750, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-20-0.69.keras

Epoch 21: val_accuracy improved from 0.68750 to 0.70833, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-21-0.71.keras

Epoch 22: val_accuracy improved from 0.70833 to 0.72917, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-22-0.73.keras
```

```
Epoch 23: val_accuracy did not improve from 0.72917

Epoch 24: val_accuracy did not improve from 0.72917

Epoch 25: val_accuracy did not improve from 0.72917

Epoch 26: val_accuracy did not improve from 0.72917

Epoch 27: val_accuracy did not improve from 0.72917

Epoch 28: val_accuracy did not improve from 0.72917

Epoch 29: val_accuracy did not improve from 0.72917

Epoch 30: val_accuracy did not improve from 0.72917

Epoch 31: val_accuracy did not improve from 0.72917

Epoch 32: val_accuracy did not improve from 0.72917

Epoch 33: val_accuracy did not improve from 0.72917

Epoch 34: val_accuracy did not improve from 0.72917

Epoch 35: val_accuracy did not improve from 0.72917

Epoch 36: val_accuracy did not improve from 0.72917

Epoch 37: val_accuracy did not improve from 0.72917

Epoch 38: val_accuracy did not improve from 0.72917

Epoch 39: val_accuracy did not improve from 0.72917

Epoch 40: val_accuracy did not improve from 0.72917

Epoch 41: val_accuracy did not improve from 0.72917

Epoch 42: val_accuracy did not improve from 0.72917

Epoch 43: val_accuracy did not improve from 0.72917

Epoch 44: val_accuracy did not improve from 0.72917

Epoch 45: val_accuracy did not improve from 0.72917

Epoch 46: val_accuracy did not improve from 0.72917
```

```
Epoch 47: val_accuracy did not improve from 0.72917

Epoch 48: val_accuracy did not improve from 0.72917

Epoch 49: val_accuracy did not improve from 0.72917

Epoch 50: val_accuracy did not improve from 0.72917

Epoch 51: val_accuracy did not improve from 0.72917

Epoch 52: val_accuracy did not improve from 0.72917

Epoch 53: val_accuracy did not improve from 0.72917

Epoch 54: val_accuracy did not improve from 0.72917

Epoch 55: val_accuracy did not improve from 0.72917

Epoch 56: val_accuracy did not improve from 0.72917

Epoch 57: val_accuracy did not improve from 0.72917

Epoch 58: val_accuracy did not improve from 0.72917

Epoch 59: val_accuracy did not improve from 0.72917

Epoch 60: val_accuracy did not improve from 0.72917

Epoch 61: val_accuracy did not improve from 0.72917

Epoch 62: val_accuracy did not improve from 0.72917

Epoch 63: val_accuracy did not improve from 0.72917

Epoch 64: val_accuracy did not improve from 0.72917

Epoch 65: val_accuracy did not improve from 0.72917

Epoch 66: val_accuracy did not improve from 0.72917

Epoch 67: val_accuracy did not improve from 0.72917

Epoch 68: val_accuracy did not improve from 0.72917

Epoch 69: val_accuracy did not improve from 0.72917

Epoch 70: val_accuracy did not improve from 0.72917
```

```
Epoch 71: val_accuracy did not improve from 0.72917

Epoch 72: val_accuracy did not improve from 0.72917

Epoch 73: val_accuracy did not improve from 0.72917

Epoch 74: val_accuracy did not improve from 0.72917

Epoch 75: val_accuracy did not improve from 0.72917

Epoch 76: val_accuracy did not improve from 0.72917

Epoch 77: val_accuracy did not improve from 0.72917

Epoch 78: val_accuracy did not improve from 0.72917

Epoch 79: val_accuracy did not improve from 0.72917

Epoch 80: val_accuracy did not improve from 0.72917

Epoch 81: val_accuracy did not improve from 0.72917

Epoch 82: val_accuracy did not improve from 0.72917

Epoch 83: val_accuracy did not improve from 0.72917

Epoch 84: val_accuracy did not improve from 0.72917

Epoch 85: val_accuracy did not improve from 0.72917

Epoch 86: val_accuracy did not improve from 0.72917

Epoch 87: val_accuracy did not improve from 0.72917

Epoch 88: val_accuracy did not improve from 0.72917

Epoch 89: val_accuracy did not improve from 0.72917

Epoch 90: val_accuracy did not improve from 0.72917

Epoch 91: val_accuracy did not improve from 0.72917

Epoch 92: val_accuracy did not improve from 0.72917

Epoch 93: val_accuracy did not improve from 0.72917

Epoch 94: val_accuracy did not improve from 0.72917
```

```
Epoch 95: val_accuracy did not improve from 0.72917

Epoch 96: val_accuracy did not improve from 0.72917

Epoch 97: val_accuracy did not improve from 0.72917

Epoch 98: val_accuracy did not improve from 0.72917

Epoch 99: val_accuracy did not improve from 0.72917

Epoch 100: val_accuracy did not improve from 0.72917

Epoch 101: val_accuracy did not improve from 0.72917

Epoch 102: val_accuracy did not improve from 0.72917

Epoch 103: val_accuracy did not improve from 0.72917

Epoch 104: val_accuracy did not improve from 0.72917

Epoch 105: val_accuracy did not improve from 0.72917

Epoch 106: val_accuracy did not improve from 0.72917

Epoch 107: val_accuracy did not improve from 0.72917

Epoch 108: val_accuracy did not improve from 0.72917

Epoch 109: val_accuracy did not improve from 0.72917

Epoch 110: val_accuracy did not improve from 0.72917

Epoch 111: val_accuracy did not improve from 0.72917

Epoch 112: val_accuracy did not improve from 0.72917

Epoch 113: val_accuracy did not improve from 0.72917

Epoch 114: val_accuracy did not improve from 0.72917

Epoch 115: val_accuracy did not improve from 0.72917

Epoch 116: val_accuracy did not improve from 0.72917

Epoch 117: val_accuracy did not improve from 0.72917

Epoch 118: val_accuracy did not improve from 0.72917
```

```
Epoch 119: val_accuracy did not improve from 0.72917

Epoch 120: val_accuracy improved from 0.72917 to 0.75000, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save
model/weights-improvement-120-0.75.keras

Epoch 121: val_accuracy did not improve from 0.75000

Epoch 122: val_accuracy did not improve from 0.75000

Epoch 123: val_accuracy did not improve from 0.75000

Epoch 124: val_accuracy did not improve from 0.75000

Epoch 125: val_accuracy did not improve from 0.75000

Epoch 126: val_accuracy did not improve from 0.75000

Epoch 127: val_accuracy did not improve from 0.75000

Epoch 128: val_accuracy did not improve from 0.75000

Epoch 129: val_accuracy did not improve from 0.75000

Epoch 130: val_accuracy did not improve from 0.75000

Epoch 131: val_accuracy did not improve from 0.75000

Epoch 132: val_accuracy did not improve from 0.75000

Epoch 133: val_accuracy did not improve from 0.75000

Epoch 134: val_accuracy did not improve from 0.75000

Epoch 135: val_accuracy did not improve from 0.75000

Epoch 136: val_accuracy did not improve from 0.75000

Epoch 137: val_accuracy did not improve from 0.75000

Epoch 138: val_accuracy did not improve from 0.75000

Epoch 139: val_accuracy did not improve from 0.75000

Epoch 140: val_accuracy did not improve from 0.75000

Epoch 141: val_accuracy did not improve from 0.75000
```

```
Epoch 142: val_accuracy improved from 0.75000 to 0.77083, saving model to /content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Improve save model/weights-improvement-142-0.77.keras

Epoch 143: val_accuracy did not improve from 0.77083

Epoch 144: val_accuracy did not improve from 0.77083

Epoch 145: val_accuracy did not improve from 0.77083

Epoch 146: val_accuracy did not improve from 0.77083

Epoch 147: val_accuracy did not improve from 0.77083

Epoch 148: val_accuracy did not improve from 0.77083

Epoch 149: val_accuracy did not improve from 0.77083

Epoch 150: val_accuracy did not improve from 0.77083
```

Out[ ]:

```
<keras.src.callbacks.history.History at 0x7c7dabc736a0>
```

Observed: In this part of code, there is a certain part that is only saved not all of them, that is because the it only save the improvement of the model which is great when collecting good model.

Checkpoint Best Neural Network Model only¶

In [ ]:

```
Model = Sequential()
Model.add(Dense(64, input_dim=10, activation='relu'))
Model.add(Dense(32, activation='relu'))
Model.add(Dense(6, activation='softmax'))

Model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [ ]:

```
Filepath = "/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best Model/weights.best.keras"
CheckPoint = ModelCheckpoint(filepath=Filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
Callbacks_list = [CheckPoint]
Model.fit(X_train, y_train, validation_split=0.33, epochs=150, batch_size=1000, callbacks=Callbacks_list, verbose=0)
```

```
Epoch 1: val_accuracy improved from -inf to 0.10417, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 2: val_accuracy improved from 0.10417 to 0.14583, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 3: val_accuracy improved from 0.14583 to 0.16667, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 4: val_accuracy improved from 0.16667 to 0.18750, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 5: val_accuracy improved from 0.18750 to 0.25000, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 6: val_accuracy did not improve from 0.25000

Epoch 7: val_accuracy improved from 0.25000 to 0.29167, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 8: val_accuracy did not improve from 0.29167

Epoch 9: val_accuracy improved from 0.29167 to 0.37500, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 10: val_accuracy improved from 0.37500 to 0.43750, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 11: val_accuracy did not improve from 0.43750

Epoch 12: val_accuracy improved from 0.43750 to 0.45833, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 13: val_accuracy did not improve from 0.45833

Epoch 14: val_accuracy did not improve from 0.45833

Epoch 15: val_accuracy did not improve from 0.45833
```

```
Epoch 16: val_accuracy did not improve from 0.45833

Epoch 17: val_accuracy did not improve from 0.45833

Epoch 18: val_accuracy did not improve from 0.45833

Epoch 19: val_accuracy did not improve from 0.45833

Epoch 20: val_accuracy did not improve from 0.45833

Epoch 21: val_accuracy did not improve from 0.45833

Epoch 22: val_accuracy improved from 0.45833 to 0.47917, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 23: val_accuracy did not improve from 0.47917

Epoch 24: val_accuracy did not improve from 0.47917

Epoch 25: val_accuracy did not improve from 0.47917

Epoch 26: val_accuracy did not improve from 0.47917

Epoch 27: val_accuracy improved from 0.47917 to 0.50000, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 28: val_accuracy did not improve from 0.50000

Epoch 29: val_accuracy did not improve from 0.50000

Epoch 30: val_accuracy did not improve from 0.50000

Epoch 31: val_accuracy did not improve from 0.50000

Epoch 32: val_accuracy did not improve from 0.50000

Epoch 33: val_accuracy did not improve from 0.50000

Epoch 34: val_accuracy improved from 0.50000 to 0.52083, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 35: val_accuracy did not improve from 0.52083

Epoch 36: val_accuracy improved from 0.52083 to 0.54167, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
```

```
Model/weights.best.keras

Epoch 37: val_accuracy did not improve from 0.54167

Epoch 38: val_accuracy did not improve from 0.54167

Epoch 39: val_accuracy improved from 0.54167 to 0.58333, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 40: val_accuracy did not improve from 0.58333

Epoch 41: val_accuracy did not improve from 0.58333

Epoch 42: val_accuracy did not improve from 0.58333

Epoch 43: val_accuracy did not improve from 0.58333

Epoch 44: val_accuracy did not improve from 0.58333

Epoch 45: val_accuracy did not improve from 0.58333

Epoch 46: val_accuracy did not improve from 0.58333

Epoch 47: val_accuracy did not improve from 0.58333

Epoch 48: val_accuracy did not improve from 0.58333

Epoch 49: val_accuracy improved from 0.58333 to 0.60417, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 50: val_accuracy improved from 0.60417 to 0.62500, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 51: val_accuracy did not improve from 0.62500

Epoch 52: val_accuracy did not improve from 0.62500

Epoch 53: val_accuracy did not improve from 0.62500

Epoch 54: val_accuracy did not improve from 0.62500

Epoch 55: val_accuracy did not improve from 0.62500

Epoch 56: val_accuracy did not improve from 0.62500
```

```
Epoch 57: val_accuracy improved from 0.62500 to 0.64583, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 58: val_accuracy improved from 0.64583 to 0.66667, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 59: val_accuracy did not improve from 0.66667

Epoch 60: val_accuracy improved from 0.66667 to 0.68750, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 61: val_accuracy did not improve from 0.68750

Epoch 62: val_accuracy did not improve from 0.68750

Epoch 63: val_accuracy improved from 0.68750 to 0.70833, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 64: val_accuracy did not improve from 0.70833

Epoch 65: val_accuracy improved from 0.70833 to 0.75000, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 66: val_accuracy did not improve from 0.75000

Epoch 67: val_accuracy improved from 0.75000 to 0.77083, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 68: val_accuracy did not improve from 0.77083

Epoch 69: val_accuracy did not improve from 0.77083

Epoch 70: val_accuracy did not improve from 0.77083

Epoch 71: val_accuracy did not improve from 0.77083

Epoch 72: val_accuracy did not improve from 0.77083

Epoch 73: val_accuracy did not improve from 0.77083

Epoch 74: val_accuracy did not improve from 0.77083
```

```
Epoch 75: val_accuracy did not improve from 0.77083

Epoch 76: val_accuracy did not improve from 0.77083

Epoch 77: val_accuracy did not improve from 0.77083

Epoch 78: val_accuracy did not improve from 0.77083

Epoch 79: val_accuracy did not improve from 0.77083

Epoch 80: val_accuracy did not improve from 0.77083

Epoch 81: val_accuracy did not improve from 0.77083

Epoch 82: val_accuracy did not improve from 0.77083

Epoch 83: val_accuracy did not improve from 0.77083

Epoch 84: val_accuracy did not improve from 0.77083

Epoch 85: val_accuracy did not improve from 0.77083

Epoch 86: val_accuracy did not improve from 0.77083

Epoch 87: val_accuracy did not improve from 0.77083

Epoch 88: val_accuracy did not improve from 0.77083

Epoch 89: val_accuracy did not improve from 0.77083

Epoch 90: val_accuracy did not improve from 0.77083

Epoch 91: val_accuracy did not improve from 0.77083

Epoch 92: val_accuracy did not improve from 0.77083

Epoch 93: val_accuracy did not improve from 0.77083

Epoch 94: val_accuracy did not improve from 0.77083

Epoch 95: val_accuracy improved from 0.77083 to 0.79167, saving model to
/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras

Epoch 96: val_accuracy did not improve from 0.79167

Epoch 97: val_accuracy did not improve from 0.79167
```

```
Epoch 98: val_accuracy did not improve from 0.79167

Epoch 99: val_accuracy did not improve from 0.79167

Epoch 100: val_accuracy did not improve from 0.79167

Epoch 101: val_accuracy did not improve from 0.79167

Epoch 102: val_accuracy did not improve from 0.79167

Epoch 103: val_accuracy did not improve from 0.79167

Epoch 104: val_accuracy did not improve from 0.79167

Epoch 105: val_accuracy did not improve from 0.79167

Epoch 106: val_accuracy did not improve from 0.79167

Epoch 107: val_accuracy did not improve from 0.79167

Epoch 108: val_accuracy did not improve from 0.79167

Epoch 109: val_accuracy did not improve from 0.79167

Epoch 110: val_accuracy did not improve from 0.79167

Epoch 111: val_accuracy did not improve from 0.79167

Epoch 112: val_accuracy did not improve from 0.79167

Epoch 113: val_accuracy did not improve from 0.79167

Epoch 114: val_accuracy did not improve from 0.79167

Epoch 115: val_accuracy did not improve from 0.79167

Epoch 116: val_accuracy did not improve from 0.79167

Epoch 117: val_accuracy did not improve from 0.79167

Epoch 118: val_accuracy did not improve from 0.79167

Epoch 119: val_accuracy did not improve from 0.79167

Epoch 120: val_accuracy did not improve from 0.79167

Epoch 121: val_accuracy did not improve from 0.79167
```

```
Epoch 122: val_accuracy did not improve from 0.79167

Epoch 123: val_accuracy did not improve from 0.79167

Epoch 124: val_accuracy did not improve from 0.79167

Epoch 125: val_accuracy did not improve from 0.79167

Epoch 126: val_accuracy did not improve from 0.79167

Epoch 127: val_accuracy did not improve from 0.79167

Epoch 128: val_accuracy did not improve from 0.79167

Epoch 129: val_accuracy did not improve from 0.79167

Epoch 130: val_accuracy did not improve from 0.79167

Epoch 131: val_accuracy did not improve from 0.79167

Epoch 132: val_accuracy did not improve from 0.79167

Epoch 133: val_accuracy did not improve from 0.79167

Epoch 134: val_accuracy did not improve from 0.79167

Epoch 135: val_accuracy did not improve from 0.79167

Epoch 136: val_accuracy did not improve from 0.79167

Epoch 137: val_accuracy did not improve from 0.79167

Epoch 138: val_accuracy did not improve from 0.79167

Epoch 139: val_accuracy did not improve from 0.79167

Epoch 140: val_accuracy did not improve from 0.79167

Epoch 141: val_accuracy did not improve from 0.79167

Epoch 142: val_accuracy did not improve from 0.79167

Epoch 143: val_accuracy did not improve from 0.79167

Epoch 144: val_accuracy did not improve from 0.79167

Epoch 145: val_accuracy did not improve from 0.79167
```

```
Epoch 146: val_accuracy did not improve from 0.79167

Epoch 147: val_accuracy did not improve from 0.79167

Epoch 148: val_accuracy did not improve from 0.79167

Epoch 149: val_accuracy did not improve from 0.79167

Epoch 150: val_accuracy did not improve from 0.79167
```

Out[ ]:

```
<keras.src.callbacks.history.History at 0x7c7db4b6ae90>
```

Observation: In this part of code, it does the same thing as the code above, the only different is that it replace or being overlaid the previous improvement and save the best model that is being recorded

Load a saved Neural Network model¶

In [ ]:

```
Model = Sequential()
Model.add(Dense(64, input_dim=10, activation='relu'))
Model.add(Dense(32, activation='relu'))
Model.add(Dense(6, activation='softmax'))

Model.load_weights("/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/Save Best
Model/weights.best.keras")
Model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
print("Created model weights from file")and loaded
```

```
Created model weights from file
```

In [ ]:

```
Result = Model.evaluate(X_train, y_train, verbose = 0)
print("%s: %.2f%%" % (Model.metrics_names[1], Result[1]*100))
```

```
compile_metrics: 89.51%
```

Visualize Model Training History in Keras¶

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np
```

```
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical

Data = pd.read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/glass_data_with_header.csv')
Data
```

Out[ ]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [ ]:

```
X = Data.iloc[:, :-1]
y = Data.iloc[:, -1]
```

```
SS = StandardScaler()
X = SS.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=123)

LE = LabelEncoder()
y_train = LE.fit_transform(y_train)
y_test = LE.fit_transform(y_test)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [ ]:

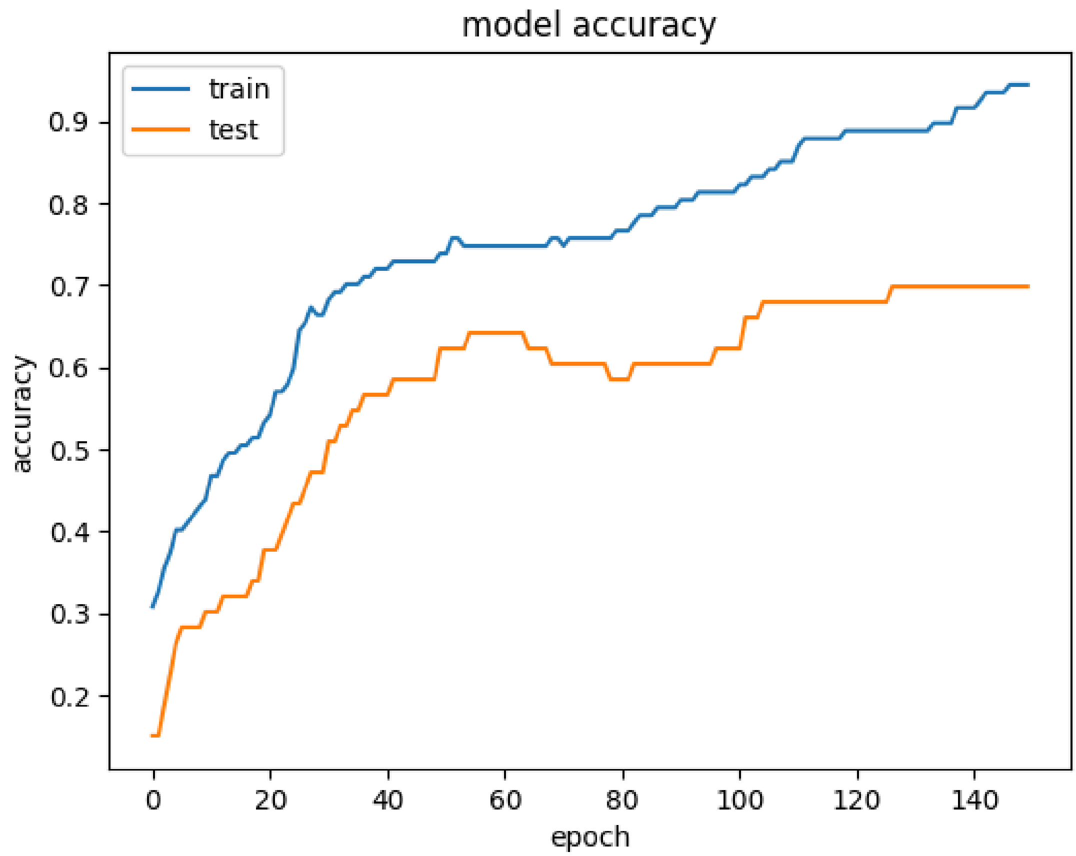
```
Model = Sequential()
Model.add(Dense(32, input_dim=10, activation='relu'))
Model.add(Dense(16, activation='relu'))
Model.add(Dense(6, activation='softmax'))

Model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
history = Model.fit(X_train, y_train, validation_split=0.33, epochs=150,
batch_size=1000, verbose=0)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

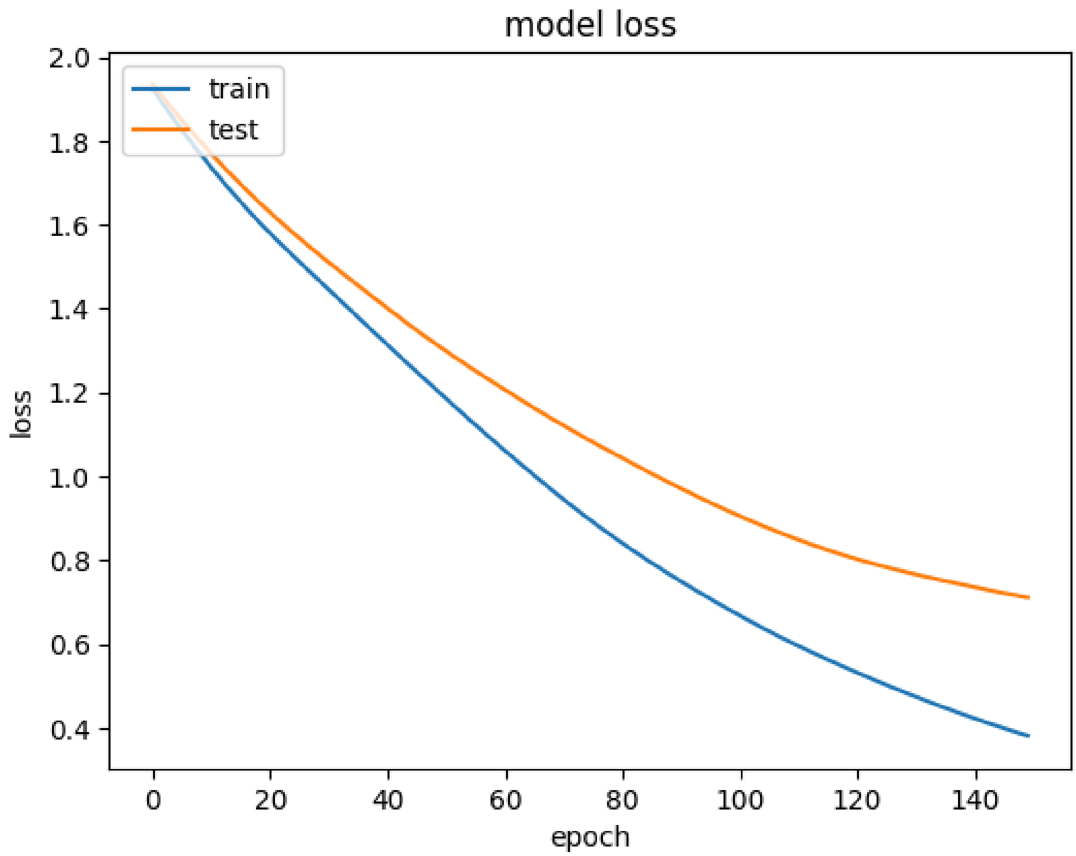
In [ ]:

```
print(history.history.keys())
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
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()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```



model accuracy
train
test
accuracy
epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100
120
140



Observation: As seen in the plot, there is still a gap between 2 plots, the train and test is going great but not enough due the gap of the Train and test is still creating gap by time.

Show the application of Dropout Regularization¶

In [ ]:

```
!pip install scikeras
```

```
Requirement already satisfied: scikeras in
/usr/local/lib/python3.10/dist-packages (0.13.0)
Requirement already satisfied: keras>=3.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (3.4.1)
Requirement already satisfied: scikit-learn>=1.4.2 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (1.5.1)
Requirement already satisfied: absl-py in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (1.4.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(1.25.2)
Requirement already satisfied: rich in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(13.7.1)
Requirement already satisfied: namex in
```

```
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (0.0.8)
Requirement already satisfied: h5py in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (3.9.0)
Requirement already satisfied: optree in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras)
(0.11.0)
Requirement already satisfied: ml-dtypes in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (0.2.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->scikeras) (24.1)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras)
(1.11.4)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras)
(1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.4.2->scikeras)
(3.5.0)
Requirement already satisfied: typing-extensions>=4.0.0 in
/usr/local/lib/python3.10/dist-packages (from optree->keras>=3.2.0->scikeras)
(4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->scikeras)
(2.16.1)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.10/dist-packages (from
markdown-it-py>=2.2.0->rich->keras>=3.2.0->scikeras) (0.1.2)
```

In [ ]:

```
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dropout
from tensorflow.keras.constraints import MaxNorm
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

Data = read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA
```

```
8.1/glass_data_with_header.csv')
Data
```

Out[ ]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [ ]:

```
X = Data.iloc[:, :-1]
y = Data.iloc[:, -1]

SS = StandardScaler()
X = SS.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=123)

X = X_train
y = y_train
```

```
LE = LabelEncoder()
y = LE.fit_transform(y)
```

In [ ]:

```
def create_baseline():
    model = Sequential()
    model.add(Dense(64, input_shape=(10,), activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(learning_rate=0.01, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
    return model
```

In [ ]:

```
Estimators = []
Estimators.append(('standardize', StandardScaler()))
Estimators.append(('mlp', KerasClassifier(model = create_baseline,
epochs=300, batch_size=1000, verbose=0)))
pipeline = Pipeline(Estimators)
Kfold = StratifiedKFold(n_splits=10, shuffle=True)
Results = cross_val_score(pipeline, X, y, cv=Kfold)
print("Baseline: %.2f%% (%.2f%%)" % (Results.mean()*100, Results.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776
: UserWarning: The least populated class in y has only 7 members, which is
less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Baseline: 36.25% (2.50%)
```

Observation: In this part of the code, the use of Dropout Regularzation is good thing when preprocessing data, because not only reducing the chances or reducing of being overfitting, it is also avoiding removing data that might contain some important part.

Show the application of Dropout on the visible layer¶

In [ ]:

```
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dropout
from tensorflow.keras.constraints import MaxNorm
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

Data = read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/glass_data_with_header.csv')
Data
```

Out[ ]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [ ]:

```
X = Data.iloc[:, :-1]
y = Data.iloc[:, -1]

SS = StandardScaler()
X = SS.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=123)

LE = LabelEncoder()
y_train = LE.fit_transform(y_train)
#y_test = LE.fit_transform(y)
```

In [ ]:

```
print(X_train.shape)
print(y_train)
```

```
(160, 10)
[0 0 1 1 4 0 5 3 0 0 2 0 1 3 1 4 2 5 3 1 1 0 3 1 5 0 0 0 0 5 1 1 0 4 3 1 0
 0 0 5 1 1 0 0 0 1 0 2 0 0 2 0 1 1 0 0 2 4 0 1 4 1 1 0 5 0 0 0 1 2 1 5 4 2
 1 0 0 1 1 5 2 0 0 1 2 0 1 0 0 0 0 5 0 0 1 0 1 1 0 1 1 5 1 3 0 2 0 1 0 2 1
 0 0 5 1 3 1 0 2 1 1 1 1 1 1 1 1 1 4 5 5 5 0 1 0 0 0 0 5 3 1 5 2 1 3 0 1 0
 5 1 1 0 1 1 1 0 1 0 1 1]
```

In [ ]:

```
def create_baseline():
    model = Sequential()
    model.add(Dropout(0.5, input_shape = (10,)))
    model.add(Dense(32, activation='relu', kernel_constraint = MaxNorm(3)))
    model.add(Dense(16, activation='relu', kernel_constraint = MaxNorm(3)))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(learning_rate=0.1, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
    return model
```

In [ ]:

```
Estimators = []
Estimators.append(('standardize', StandardScaler()))
Estimators.append(('mlp', KerasClassifier(model = create_baseline,
epochs=150, batch_size=50, verbose=0)))
pipeline = Pipeline(Estimators)
Kfold = StratifiedKFold(n_splits=10, shuffle=True)
RResults = cross_val_score(pipeline, X_train, y_train, cv=Kfold)
print("Visible: %.2f%% (%.2f%%)" % (RResults.mean()*100, RResults.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776
: UserWarning: The least populated class in y has only 7 members, which is
less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
```

```
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/regularization/dropo
ut.py:42: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)

Visible: 36.25% (2.50%)
```

In [ ]:

```
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dropout
from tensorflow.keras.constraints import MaxNorm
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline

Data = read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA
8.1/glass_data_with_header.csv')
Data
```

Out[ ]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7

213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [ ]:

```
X = Data.iloc[:, :-1]
y = Data.iloc[:, -1]

SS = StandardScaler()
X = SS.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=123)

LE = LabelEncoder()
y_encoded = LE.fit_transform(y_train)
#y_test = LE.fit_transform(y)
```

In [ ]:

```
def create_baseline():
    model = Sequential()
    model.add(Dense(32, input_shape = (10,), activation = 'relu',
kernel_constraint = MaxNorm(3)))
    model.add(Dropout(0.5))
    model.add(Dense(16, activation='relu', kernel_constraint = MaxNorm(3)))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    sgd = SGD(learning_rate=0.0001, momentum=0.8)
    model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
    return model
```

In [ ]:

```
Estimators = []
Estimators.append(('standardize', StandardScaler()))
Estimators.append(('mlp', KerasClassifier(model = create_baseline,
epochs=300, batch_size=50, verbose=0)))
pipeline = Pipeline(Estimators)
Kfold = StratifiedKFold(n_splits=10, shuffle=True)
HResults = cross_val_score(pipeline, X_train, y_encoded, cv=Kfold)
print("Hidden: %.2f%% (%.2f%%)" % (HResults.mean()*100, HResults.std()*100))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:776
: UserWarning: The least populated class in y has only 7 members, which is
less than n_splits=10.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
```

```
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Hidden: 36.25% (2.50%)
```

Observation: In this part of the code, both Visible and Hidden layer has the same percentages, although the process is different.

Show the application of a time-based learning rate schedule¶

In [ ]:

```
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# tf.keras.optimizers.legacy.SGD
from tensorflow.keras.optimizers.legacy import SGD
from sklearn.preprocessing import LabelEncoder
```

In [ ]:

```
Data = read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/glass_data_with_header.csv')
Data
```

Out[ ]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7

213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [ ]:

```
Dataset = Data.values
Dataset
```

Out[ ]:

```
array([[  1.     ,   1.52101,  13.64   , ...,   0.     ,   0.     ,
          1.     ],
       [  2.     ,   1.51761,  13.89   , ...,   0.     ,   0.     ,
          1.     ],
       [  3.     ,   1.51618,  13.53   , ...,   0.     ,   0.     ,
          1.     ],
       ...,
       [212.     ,   1.52065,  14.36   , ...,   1.64   ,   0.     ,
          7.     ],
       [213.     ,   1.51651,  14.38   , ...,   1.57   ,   0.     ,
          7.     ],
       [214.     ,   1.51711,  14.23   , ...,   1.67   ,   0.     ,
          7.     ]])
```

In [ ]:

```
X = Dataset[:, :-1]
y = Dataset[:, -1]
```

In [ ]:

```
LE = LabelEncoder()
y = LE.fit_transform(y)
```

In [ ]:

```
Model = Sequential()
Model.add(Dense(32, input_shape = (10,), activation='relu'))
Model.add(Dense(16, activation='relu'))
Model.add(Dense(1, activation='sigmoid'))

Epochs = 150
learning_rate = 0.1
decay_rate = learning_rate / Epochs
momentum = 0.8
sgd = SGD(learning_rate=learning_rate, momentum=momentum, decay=decay_rate,
nesterov=False)
Model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
```

```
Model.fit(X, y, validation_split = 0.33, epochs=Epochs, batch_size=500,
verbose=2)
```

```
Epoch 1/150
1/1 - 2s - loss: 0.3530 - accuracy: 0.8322 - val_loss: 617.8101 -
val_accuracy: 0.0000e+00 - 2s/epoch - 2s/step
Epoch 2/150
1/1 - 0s - loss: 65.6476 - accuracy: 0.4895 - val_loss: 148.8981 -
val_accuracy: 0.0000e+00 - 180ms/epoch - 180ms/step
Epoch 3/150
1/1 - 0s - loss: 13.0404 - accuracy: 0.4476 - val_loss: -4.4593e+04 -
val_accuracy: 0.0423 - 78ms/epoch - 78ms/step
Epoch 4/150
1/1 - 0s - loss: 2506.6309 - accuracy: 0.5105 - val_loss: -1.1996e+04 -
val_accuracy: 0.0423 - 103ms/epoch - 103ms/step
Epoch 5/150
1/1 - 0s - loss: 1313.3936 - accuracy: 0.5105 - val_loss: 51139.6758 -
val_accuracy: 0.0000e+00 - 70ms/epoch - 70ms/step
Epoch 6/150
1/1 - 0s - loss: 5660.3306 - accuracy: 0.4895 - val_loss: -4.4740e+01 -
val_accuracy: 0.0423 - 103ms/epoch - 103ms/step
Epoch 7/150
1/1 - 0s - loss: 8.3599 - accuracy: 0.5105 - val_loss: 0.5548 - val_accuracy:
0.0423 - 83ms/epoch - 83ms/step
Epoch 8/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.6717 - val_accuracy:
0.0423 - 52ms/epoch - 52ms/step
Epoch 9/150
1/1 - 0s - loss: 0.6931 - accuracy: 0.5105 - val_loss: 0.7619 - val_accuracy:
0.0000e+00 - 61ms/epoch - 61ms/step
Epoch 10/150
1/1 - 0s - loss: 0.6934 - accuracy: 0.4895 - val_loss: 0.8279 - val_accuracy:
0.0000e+00 - 79ms/epoch - 79ms/step
Epoch 11/150
1/1 - 0s - loss: 0.6938 - accuracy: 0.4895 - val_loss: 0.8722 - val_accuracy:
0.0000e+00 - 63ms/epoch - 63ms/step
Epoch 12/150
1/1 - 0s - loss: 0.6942 - accuracy: 0.4895 - val_loss: 0.8976 - val_accuracy:
0.0000e+00 - 77ms/epoch - 77ms/step
Epoch 13/150
1/1 - 0s - loss: 0.6944 - accuracy: 0.4895 - val_loss: 0.9071 - val_accuracy:
0.0000e+00 - 86ms/epoch - 86ms/step
Epoch 14/150
1/1 - 0s - loss: 0.6944 - accuracy: 0.4895 - val_loss: 0.9036 - val_accuracy:
0.0000e+00 - 81ms/epoch - 81ms/step
Epoch 15/150
1/1 - 0s - loss: 0.6944 - accuracy: 0.4895 - val_loss: 0.8897 - val_accuracy:
0.0000e+00 - 83ms/epoch - 83ms/step
Epoch 16/150
1/1 - 0s - loss: 0.6943 - accuracy: 0.4895 - val_loss: 0.8681 - val_accuracy:
```

```
0.0000e+00 - 62ms/epoch - 62ms/step
Epoch 17/150
1/1 - 0s - loss: 0.6941 - accuracy: 0.4895 - val_loss: 0.8411 - val_accuracy: 
0.0000e+00 - 65ms/epoch - 65ms/step
Epoch 18/150
1/1 - 0s - loss: 0.6939 - accuracy: 0.4895 - val_loss: 0.8105 - val_accuracy: 
0.0000e+00 - 88ms/epoch - 88ms/step
Epoch 19/150
1/1 - 0s - loss: 0.6937 - accuracy: 0.4895 - val_loss: 0.7782 - val_accuracy: 
0.0000e+00 - 73ms/epoch - 73ms/step
Epoch 20/150
1/1 - 0s - loss: 0.6935 - accuracy: 0.4895 - val_loss: 0.7454 - val_accuracy: 
0.0000e+00 - 72ms/epoch - 72ms/step
Epoch 21/150
1/1 - 0s - loss: 0.6934 - accuracy: 0.4895 - val_loss: 0.7135 - val_accuracy: 
0.0000e+00 - 71ms/epoch - 71ms/step
Epoch 22/150
1/1 - 0s - loss: 0.6932 - accuracy: 0.4895 - val_loss: 0.6831 - val_accuracy: 
0.0423 - 74ms/epoch - 74ms/step
Epoch 23/150
1/1 - 0s - loss: 0.6931 - accuracy: 0.5105 - val_loss: 0.6551 - val_accuracy: 
0.0423 - 74ms/epoch - 74ms/step
Epoch 24/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6297 - val_accuracy: 
0.0423 - 70ms/epoch - 70ms/step
Epoch 25/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6073 - val_accuracy: 
0.0423 - 66ms/epoch - 66ms/step
Epoch 26/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.5880 - val_accuracy: 
0.0423 - 77ms/epoch - 77ms/step
Epoch 27/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5718 - val_accuracy: 
0.0423 - 117ms/epoch - 117ms/step
Epoch 28/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5585 - val_accuracy: 
0.0423 - 32ms/epoch - 32ms/step
Epoch 29/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5480 - val_accuracy: 
0.0423 - 33ms/epoch - 33ms/step
Epoch 30/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5401 - val_accuracy: 
0.0423 - 31ms/epoch - 31ms/step
Epoch 31/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5344 - val_accuracy: 
0.0423 - 33ms/epoch - 33ms/step
Epoch 32/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5308 - val_accuracy: 
0.0423 - 37ms/epoch - 37ms/step
Epoch 33/150
```

```
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5289 - val_accuracy:
0.0423 - 36ms/epoch - 36ms/step
Epoch 34/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5284 - val_accuracy:
0.0423 - 40ms/epoch - 40ms/step
Epoch 35/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5291 - val_accuracy:
0.0423 - 42ms/epoch - 42ms/step
Epoch 36/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5306 - val_accuracy:
0.0423 - 56ms/epoch - 56ms/step
Epoch 37/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5329 - val_accuracy:
0.0423 - 35ms/epoch - 35ms/step
Epoch 38/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5356 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 39/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5387 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 40/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5418 - val_accuracy:
0.0423 - 33ms/epoch - 33ms/step
Epoch 41/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5450 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 42/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5481 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 43/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5510 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 44/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5537 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 45/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5561 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 46/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5582 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 47/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5601 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 48/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5616 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 49/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5629 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
```

```
Epoch 50/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5639 - val_accuracy:
0.0423 - 57ms/epoch - 57ms/step
Epoch 51/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5646 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 52/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5652 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 53/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5655 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 54/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5657 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 55/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5657 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 56/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5657 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 57/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5655 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 58/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5653 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 59/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5650 - val_accuracy:
0.0423 - 33ms/epoch - 33ms/step
Epoch 60/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5648 - val_accuracy:
0.0423 - 34ms/epoch - 34ms/step
Epoch 61/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5645 - val_accuracy:
0.0423 - 42ms/epoch - 42ms/step
Epoch 62/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5641 - val_accuracy:
0.0423 - 54ms/epoch - 54ms/step
Epoch 63/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5639 - val_accuracy:
0.0423 - 56ms/epoch - 56ms/step
Epoch 64/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5636 - val_accuracy:
0.0423 - 46ms/epoch - 46ms/step
Epoch 65/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5633 - val_accuracy:
0.0423 - 55ms/epoch - 55ms/step
Epoch 66/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5631 - val_accuracy:
```

```
0.0423 - 39ms/epoch - 39ms/step
Epoch 67/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5629 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 68/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5627 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 69/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5626 - val_accuracy:
0.0423 - 52ms/epoch - 52ms/step
Epoch 70/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 71/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5623 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 72/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5623 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 73/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5622 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 74/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5622 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 75/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5621 - val_accuracy:
0.0423 - 57ms/epoch - 57ms/step
Epoch 76/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5621 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 77/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5621 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 78/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5622 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 79/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5622 - val_accuracy:
0.0423 - 33ms/epoch - 33ms/step
Epoch 80/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5622 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 81/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5622 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 82/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5623 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 83/150
```

```
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5623 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 84/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5623 - val_accuracy:
0.0423 - 36ms/epoch - 36ms/step
Epoch 85/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5623 - val_accuracy:
0.0423 - 39ms/epoch - 39ms/step
Epoch 86/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 53ms/epoch - 53ms/step
Epoch 87/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 54ms/epoch - 54ms/step
Epoch 88/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 39ms/epoch - 39ms/step
Epoch 89/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 35ms/epoch - 35ms/step
Epoch 90/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 91/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 92/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 93/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 94/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 95/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 96/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 97/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 33ms/epoch - 33ms/step
Epoch 98/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 99/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
```

```
Epoch 100/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 37ms/epoch - 37ms/step
Epoch 101/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 102/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 45ms/epoch - 45ms/step
Epoch 103/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 104/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 105/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 106/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 33ms/epoch - 33ms/step
Epoch 107/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 29ms/epoch - 29ms/step
Epoch 108/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 48ms/epoch - 48ms/step
Epoch 109/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 54ms/epoch - 54ms/step
Epoch 110/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 37ms/epoch - 37ms/step
Epoch 111/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 56ms/epoch - 56ms/step
Epoch 112/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 53ms/epoch - 53ms/step
Epoch 113/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 36ms/epoch - 36ms/step
Epoch 114/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 41ms/epoch - 41ms/step
Epoch 115/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 53ms/epoch - 53ms/step
Epoch 116/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
```

```
0.0423 - 31ms/epoch - 31ms/step
Epoch 117/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 118/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 119/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 120/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 121/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 122/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 33ms/epoch - 33ms/step
Epoch 123/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 124/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 125/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 126/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 127/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 41ms/epoch - 41ms/step
Epoch 128/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5624 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 129/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 130/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 131/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 132/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 51ms/epoch - 51ms/step
Epoch 133/150
```

```
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 50ms/epoch - 50ms/step
Epoch 134/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 37ms/epoch - 37ms/step
Epoch 135/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 37ms/epoch - 37ms/step
Epoch 136/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 37ms/epoch - 37ms/step
Epoch 137/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 38ms/epoch - 38ms/step
Epoch 138/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 53ms/epoch - 53ms/step
Epoch 139/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 35ms/epoch - 35ms/step
Epoch 140/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 34ms/epoch - 34ms/step
Epoch 141/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 35ms/epoch - 35ms/step
Epoch 142/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 143/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 144/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 50ms/epoch - 50ms/step
Epoch 145/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 32ms/epoch - 32ms/step
Epoch 146/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 48ms/epoch - 48ms/step
Epoch 147/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 30ms/epoch - 30ms/step
Epoch 148/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 31ms/epoch - 31ms/step
Epoch 149/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy:
0.0423 - 34ms/epoch - 34ms/step
```

```
Epoch 150/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5625 - val_accuracy: 0.0423 - 31ms/epoch - 31ms/step
```

Out[ ]:

```
<keras.src.callbacks.History at 0x7cc77b3a2350>
```

Show the application of a drop-based learning rate schedule¶

In [ ]:

```
from pandas import read_csv
import math
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers.legacy import SGD
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.callbacks import LearningRateScheduler
```

In [ ]:

```
Data = read_csv('/content/drive/MyDrive/CPE 019 (Retake)/HOA 8.1/glass_data_with_header.csv')
Data
```

Out[ ]:

	Id_number	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type_of_glass
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7

211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [ ]:

```
def step_decay(epoch):
    initial_lrate = 0.1
    drop = 0.5
    epochs_drop = 10.0
    lrate = initial_lrate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate
```

In [ ]:

```
Dataset = Data.values
X = Dataset[:, :-1]
y = Dataset[:, -1]
LE = LabelEncoder()
y = LE.fit_transform(y)
```

In [ ]:

```
Model = Sequential()
Model.add(Dense(32, input_shape = (10,), activation='relu'))
Model.add(Dense(1, activation='sigmoid'))


sgd = SGD(learning_rate = 0.0, momentum = 0.9)
Model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
lrate = LearningRateScheduler(step_decay)
callbacks_list = [lrate]

Model.fit(X, y, validation_split = 0.33, epochs=150, batch_size=500,
callbacks = callbacks_list, verbose=2)
```

```
Epoch 1/150
1/1 - 1s - loss: 9.0685 - accuracy: 0.3846 - val_loss: -5.6592e+03 -
val_accuracy: 0.0423 - lr: 0.1000 - 1s/epoch - 1s/step
Epoch 2/150
1/1 - 0s - loss: 357.2803 - accuracy: 0.5105 - val_loss: 7938.7246 -
val_accuracy: 0.0000e+00 - lr: 0.1000 - 120ms/epoch - 120ms/step
Epoch 3/150
1/1 - 0s - loss: 694.6346 - accuracy: 0.4895 - val_loss: -4.8873e+01 -
```

```
val_accuracy: 0.0423 - lr: 0.1000 - 170ms/epoch - 170ms/step
Epoch 4/150
1/1 - 0s - loss: 0.3765 - accuracy: 0.8531 - val_loss: -1.2058e+00 -
val_accuracy: 0.0000e+00 - lr: 0.1000 - 188ms/epoch - 188ms/step
Epoch 5/150
1/1 - 0s - loss: 3.8208 - accuracy: 0.4895 - val_loss: -1.5414e+03 -
val_accuracy: 0.0423 - lr: 0.1000 - 76ms/epoch - 76ms/step
Epoch 6/150
1/1 - 0s - loss: 52.6741 - accuracy: 0.5105 - val_loss: 0.1847 -
val_accuracy: 0.0423 - lr: 0.1000 - 56ms/epoch - 56ms/step
Epoch 7/150
1/1 - 0s - loss: 0.6948 - accuracy: 0.5105 - val_loss: 0.1620 - val_accuracy:
0.0423 - lr: 0.1000 - 110ms/epoch - 110ms/step
Epoch 8/150
1/1 - 0s - loss: 0.6950 - accuracy: 0.5105 - val_loss: 0.1514 - val_accuracy:
0.0423 - lr: 0.1000 - 187ms/epoch - 187ms/step
Epoch 9/150
1/1 - 0s - loss: 0.6951 - accuracy: 0.5105 - val_loss: 0.1521 - val_accuracy:
0.0423 - lr: 0.1000 - 86ms/epoch - 86ms/step
Epoch 10/150
1/1 - 0s - loss: 0.6951 - accuracy: 0.5105 - val_loss: 0.1578 - val_accuracy:
0.0423 - lr: 0.0500 - 87ms/epoch - 87ms/step
Epoch 11/150
1/1 - 0s - loss: 0.6951 - accuracy: 0.5105 - val_loss: 0.1680 - val_accuracy:
0.0423 - lr: 0.0500 - 95ms/epoch - 95ms/step
Epoch 12/150
1/1 - 0s - loss: 0.6950 - accuracy: 0.5105 - val_loss: 0.1820 - val_accuracy:
0.0423 - lr: 0.0500 - 199ms/epoch - 199ms/step
Epoch 13/150
1/1 - 0s - loss: 0.6948 - accuracy: 0.5105 - val_loss: 0.1994 - val_accuracy:
0.0423 - lr: 0.0500 - 149ms/epoch - 149ms/step
Epoch 14/150
1/1 - 0s - loss: 0.6946 - accuracy: 0.5105 - val_loss: 0.2195 - val_accuracy:
0.0423 - lr: 0.0500 - 125ms/epoch - 125ms/step
Epoch 15/150
1/1 - 0s - loss: 0.6945 - accuracy: 0.5105 - val_loss: 0.2419 - val_accuracy:
0.0423 - lr: 0.0500 - 117ms/epoch - 117ms/step
Epoch 16/150
1/1 - 0s - loss: 0.6943 - accuracy: 0.5105 - val_loss: 0.2661 - val_accuracy:
0.0423 - lr: 0.0500 - 54ms/epoch - 54ms/step
Epoch 17/150
1/1 - 0s - loss: 0.6941 - accuracy: 0.5105 - val_loss: 0.2915 - val_accuracy:
0.0423 - lr: 0.0500 - 113ms/epoch - 113ms/step
Epoch 18/150
1/1 - 0s - loss: 0.6939 - accuracy: 0.5105 - val_loss: 0.3177 - val_accuracy:
0.0423 - lr: 0.0500 - 78ms/epoch - 78ms/step
Epoch 19/150
1/1 - 0s - loss: 0.6937 - accuracy: 0.5105 - val_loss: 0.3444 - val_accuracy:
0.0423 - lr: 0.0500 - 82ms/epoch - 82ms/step
Epoch 20/150
```

```
1/1 - 0s - loss: 0.6935 - accuracy: 0.5105 - val_loss: 0.3699 - val_accuracy:
0.0423 - lr: 0.0250 - 94ms/epoch - 94ms/step
Epoch 21/150
1/1 - 0s - loss: 0.6934 - accuracy: 0.5105 - val_loss: 0.3939 - val_accuracy:
0.0423 - lr: 0.0250 - 88ms/epoch - 88ms/step
Epoch 22/150
1/1 - 0s - loss: 0.6933 - accuracy: 0.5105 - val_loss: 0.4167 - val_accuracy:
0.0423 - lr: 0.0250 - 130ms/epoch - 130ms/step
Epoch 23/150
1/1 - 0s - loss: 0.6932 - accuracy: 0.5105 - val_loss: 0.4380 - val_accuracy:
0.0423 - lr: 0.0250 - 182ms/epoch - 182ms/step
Epoch 24/150
1/1 - 0s - loss: 0.6931 - accuracy: 0.5105 - val_loss: 0.4581 - val_accuracy:
0.0423 - lr: 0.0250 - 86ms/epoch - 86ms/step
Epoch 25/150
1/1 - 0s - loss: 0.6931 - accuracy: 0.5105 - val_loss: 0.4768 - val_accuracy:
0.0423 - lr: 0.0250 - 149ms/epoch - 149ms/step
Epoch 26/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.4941 - val_accuracy:
0.0423 - lr: 0.0250 - 114ms/epoch - 114ms/step
Epoch 27/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.5102 - val_accuracy:
0.0423 - lr: 0.0250 - 104ms/epoch - 104ms/step
Epoch 28/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.5250 - val_accuracy:
0.0423 - lr: 0.0250 - 122ms/epoch - 122ms/step
Epoch 29/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5385 - val_accuracy:
0.0423 - lr: 0.0250 - 71ms/epoch - 71ms/step
Epoch 30/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5508 - val_accuracy:
0.0423 - lr: 0.0125 - 123ms/epoch - 123ms/step
Epoch 31/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5619 - val_accuracy:
0.0423 - lr: 0.0125 - 80ms/epoch - 80ms/step
Epoch 32/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5718 - val_accuracy:
0.0423 - lr: 0.0125 - 122ms/epoch - 122ms/step
Epoch 33/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5808 - val_accuracy:
0.0423 - lr: 0.0125 - 87ms/epoch - 87ms/step
Epoch 34/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5888 - val_accuracy:
0.0423 - lr: 0.0125 - 67ms/epoch - 67ms/step
Epoch 35/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.5959 - val_accuracy:
0.0423 - lr: 0.0125 - 71ms/epoch - 71ms/step
Epoch 36/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.6022 - val_accuracy:
0.0423 - lr: 0.0125 - 70ms/epoch - 70ms/step
```

```
Epoch 37/150
1/1 - 0s - loss: 0.6929 - accuracy: 0.5105 - val_loss: 0.6078 - val_accuracy:
0.0423 - lr: 0.0125 - 64ms/epoch - 64ms/step
Epoch 38/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6126 - val_accuracy:
0.0423 - lr: 0.0125 - 74ms/epoch - 74ms/step
Epoch 39/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6168 - val_accuracy:
0.0423 - lr: 0.0125 - 133ms/epoch - 133ms/step
Epoch 40/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6205 - val_accuracy:
0.0423 - lr: 0.0063 - 172ms/epoch - 172ms/step
Epoch 41/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6238 - val_accuracy:
0.0423 - lr: 0.0063 - 107ms/epoch - 107ms/step
Epoch 42/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6266 - val_accuracy:
0.0423 - lr: 0.0063 - 64ms/epoch - 64ms/step
Epoch 43/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6291 - val_accuracy:
0.0423 - lr: 0.0063 - 93ms/epoch - 93ms/step
Epoch 44/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6312 - val_accuracy:
0.0423 - lr: 0.0063 - 87ms/epoch - 87ms/step
Epoch 45/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6329 - val_accuracy:
0.0423 - lr: 0.0063 - 36ms/epoch - 36ms/step
Epoch 46/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6344 - val_accuracy:
0.0423 - lr: 0.0063 - 35ms/epoch - 35ms/step
Epoch 47/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6357 - val_accuracy:
0.0423 - lr: 0.0063 - 52ms/epoch - 52ms/step
Epoch 48/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6366 - val_accuracy:
0.0423 - lr: 0.0063 - 35ms/epoch - 35ms/step
Epoch 49/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6374 - val_accuracy:
0.0423 - lr: 0.0063 - 38ms/epoch - 38ms/step
Epoch 50/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6381 - val_accuracy:
0.0423 - lr: 0.0031 - 36ms/epoch - 36ms/step
Epoch 51/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6386 - val_accuracy:
0.0423 - lr: 0.0031 - 44ms/epoch - 44ms/step
Epoch 52/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6390 - val_accuracy:
0.0423 - lr: 0.0031 - 58ms/epoch - 58ms/step
Epoch 53/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6393 - val_accuracy:
```

```
0.0423 - lr: 0.0031 - 56ms/epoch - 56ms/step
Epoch 54/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6395 - val_accuracy: 
0.0423 - lr: 0.0031 - 36ms/epoch - 36ms/step
Epoch 55/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6396 - val_accuracy: 
0.0423 - lr: 0.0031 - 51ms/epoch - 51ms/step
Epoch 56/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6397 - val_accuracy: 
0.0423 - lr: 0.0031 - 61ms/epoch - 61ms/step
Epoch 57/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6397 - val_accuracy: 
0.0423 - lr: 0.0031 - 37ms/epoch - 37ms/step
Epoch 58/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6396 - val_accuracy: 
0.0423 - lr: 0.0031 - 39ms/epoch - 39ms/step
Epoch 59/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6395 - val_accuracy: 
0.0423 - lr: 0.0031 - 32ms/epoch - 32ms/step
Epoch 60/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6394 - val_accuracy: 
0.0423 - lr: 0.0016 - 35ms/epoch - 35ms/step
Epoch 61/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6392 - val_accuracy: 
0.0423 - lr: 0.0016 - 51ms/epoch - 51ms/step
Epoch 62/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6390 - val_accuracy: 
0.0423 - lr: 0.0016 - 36ms/epoch - 36ms/step
Epoch 63/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6389 - val_accuracy: 
0.0423 - lr: 0.0016 - 32ms/epoch - 32ms/step
Epoch 64/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6387 - val_accuracy: 
0.0423 - lr: 0.0016 - 51ms/epoch - 51ms/step
Epoch 65/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6385 - val_accuracy: 
0.0423 - lr: 0.0016 - 33ms/epoch - 33ms/step
Epoch 66/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6382 - val_accuracy: 
0.0423 - lr: 0.0016 - 55ms/epoch - 55ms/step
Epoch 67/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6380 - val_accuracy: 
0.0423 - lr: 0.0016 - 34ms/epoch - 34ms/step
Epoch 68/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6378 - val_accuracy: 
0.0423 - lr: 0.0016 - 32ms/epoch - 32ms/step
Epoch 69/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6376 - val_accuracy: 
0.0423 - lr: 0.0016 - 35ms/epoch - 35ms/step
Epoch 70/150
```

```
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6373 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 32ms/epoch - 32ms/step
Epoch 71/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6371 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 52ms/epoch - 52ms/step
Epoch 72/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6369 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 42ms/epoch - 42ms/step
Epoch 73/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6367 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 43ms/epoch - 43ms/step
Epoch 74/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6365 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 40ms/epoch - 40ms/step
Epoch 75/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6363 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 56ms/epoch - 56ms/step
Epoch 76/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6361 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 41ms/epoch - 41ms/step
Epoch 77/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6359 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 31ms/epoch - 31ms/step
Epoch 78/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6358 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 54ms/epoch - 54ms/step
Epoch 79/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6356 - val_accuracy:
0.0423 - lr: 7.8125e-04 - 33ms/epoch - 33ms/step
Epoch 80/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6354 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 34ms/epoch - 34ms/step
Epoch 81/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6353 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 36ms/epoch - 36ms/step
Epoch 82/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6351 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 32ms/epoch - 32ms/step
Epoch 83/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6350 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 34ms/epoch - 34ms/step
Epoch 84/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6348 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 34ms/epoch - 34ms/step
Epoch 85/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6347 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 35ms/epoch - 35ms/step
Epoch 86/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6346 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 32ms/epoch - 32ms/step
```

```
Epoch 87/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6345 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 33ms/epoch - 33ms/step
Epoch 88/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6344 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 33ms/epoch - 33ms/step
Epoch 89/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6342 - val_accuracy:
0.0423 - lr: 3.9063e-04 - 33ms/epoch - 33ms/step
Epoch 90/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6341 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 34ms/epoch - 34ms/step
Epoch 91/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6341 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 50ms/epoch - 50ms/step
Epoch 92/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6340 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 35ms/epoch - 35ms/step
Epoch 93/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6339 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 50ms/epoch - 50ms/step
Epoch 94/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6338 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 59ms/epoch - 59ms/step
Epoch 95/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6337 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 40ms/epoch - 40ms/step
Epoch 96/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6337 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 44ms/epoch - 44ms/step
Epoch 97/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6336 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 42ms/epoch - 42ms/step
Epoch 98/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6335 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 55ms/epoch - 55ms/step
Epoch 99/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6335 - val_accuracy:
0.0423 - lr: 1.9531e-04 - 44ms/epoch - 44ms/step
Epoch 100/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6334 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 38ms/epoch - 38ms/step
Epoch 101/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6334 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 45ms/epoch - 45ms/step
Epoch 102/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6333 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 33ms/epoch - 33ms/step
Epoch 103/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6333 - val_accuracy:
```

```
0.0423 - lr: 9.7656e-05 - 33ms/epoch - 33ms/step
Epoch 104/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6332 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 33ms/epoch - 33ms/step
Epoch 105/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6332 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 50ms/epoch - 50ms/step
Epoch 106/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6332 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 36ms/epoch - 36ms/step
Epoch 107/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6331 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 33ms/epoch - 33ms/step
Epoch 108/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6331 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 51ms/epoch - 51ms/step
Epoch 109/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6330 - val_accuracy:
0.0423 - lr: 9.7656e-05 - 33ms/epoch - 33ms/step
Epoch 110/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6330 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 33ms/epoch - 33ms/step
Epoch 111/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6330 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 31ms/epoch - 31ms/step
Epoch 112/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6330 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 32ms/epoch - 32ms/step
Epoch 113/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6329 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 31ms/epoch - 31ms/step
Epoch 114/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6329 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 33ms/epoch - 33ms/step
Epoch 115/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6329 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 33ms/epoch - 33ms/step
Epoch 116/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6329 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 56ms/epoch - 56ms/step
Epoch 117/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6329 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 38ms/epoch - 38ms/step
Epoch 118/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 57ms/epoch - 57ms/step
Epoch 119/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy:
0.0423 - lr: 4.8828e-05 - 40ms/epoch - 40ms/step
Epoch 120/150
```

```
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 38ms/epoch - 38ms/step
Epoch 121/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 31ms/epoch - 31ms/step
Epoch 122/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 32ms/epoch - 32ms/step
Epoch 123/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 37ms/epoch - 37ms/step
Epoch 124/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6328 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 31ms/epoch - 31ms/step
Epoch 125/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 59ms/epoch - 59ms/step
Epoch 126/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 46ms/epoch - 46ms/step
Epoch 127/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 43ms/epoch - 43ms/step
Epoch 128/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 63ms/epoch - 63ms/step
Epoch 129/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 2.4414e-05 - 48ms/epoch - 48ms/step
Epoch 130/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 64ms/epoch - 64ms/step
Epoch 131/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 66ms/epoch - 66ms/step
Epoch 132/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 46ms/epoch - 46ms/step
Epoch 133/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 50ms/epoch - 50ms/step
Epoch 134/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 45ms/epoch - 45ms/step
Epoch 135/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 70ms/epoch - 70ms/step
Epoch 136/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy: 
0.0423 - lr: 1.2207e-05 - 78ms/epoch - 78ms/step
```

```
Epoch 137/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy:
0.0423 - lr: 1.2207e-05 - 88ms/epoch - 88ms/step
Epoch 138/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6327 - val_accuracy:
0.0423 - lr: 1.2207e-05 - 75ms/epoch - 75ms/step
Epoch 139/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 1.2207e-05 - 51ms/epoch - 51ms/step
Epoch 140/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 61ms/epoch - 61ms/step
Epoch 141/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 51ms/epoch - 51ms/step
Epoch 142/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 64ms/epoch - 64ms/step
Epoch 143/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 59ms/epoch - 59ms/step
Epoch 144/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 46ms/epoch - 46ms/step
Epoch 145/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 68ms/epoch - 68ms/step
Epoch 146/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 63ms/epoch - 63ms/step
Epoch 147/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 64ms/epoch - 64ms/step
Epoch 148/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 61ms/epoch - 61ms/step
Epoch 149/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 6.1035e-06 - 63ms/epoch - 63ms/step
Epoch 150/150
1/1 - 0s - loss: 0.6930 - accuracy: 0.5105 - val_loss: 0.6326 - val_accuracy:
0.0423 - lr: 3.0518e-06 - 63ms/epoch - 63ms/step
```

Out[ ]:

```
<keras.src.callbacks.History at 0x7dcb39be91b0>
```

Observation: In this part of the code, the time based and drop based, the two has almost has the same preocess but when it observed over time, it will shows different result, which for me the drop based is much better than time base due to being more effective learning rate adjustments.

Conclusion¶

- In this activity, I was able to learned about how to save and load models, load checkpoints and manage to improve the model. I implement a time based and drop base learning rate and used the dropout regularization to reduce overfitting. With this activity, I will be able to improve my skills in regularization, optimization, and model management for the future projects.