ICP-6

Venkatesh Spandan Kumar Saggilla
ID: 700752792

GITHUB LINK:

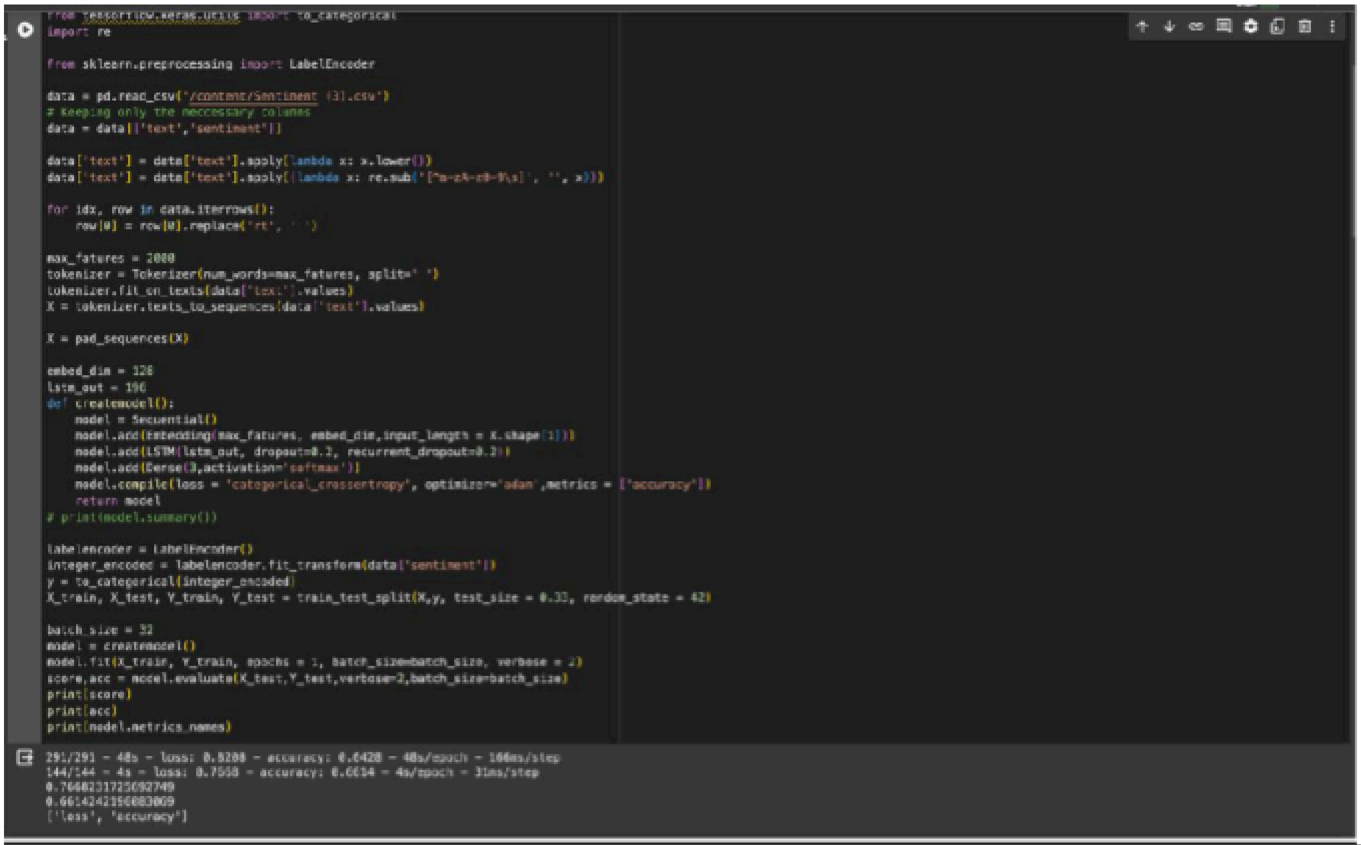
https://github.com/Venkatesh-Spandan/ICP_6_NN

VIDEO LINK

https://drive.google.com/file/d/1XXlM8vKh4g-lyShvXhiDjMEy65rZCGrJ/view?usp=drive_link

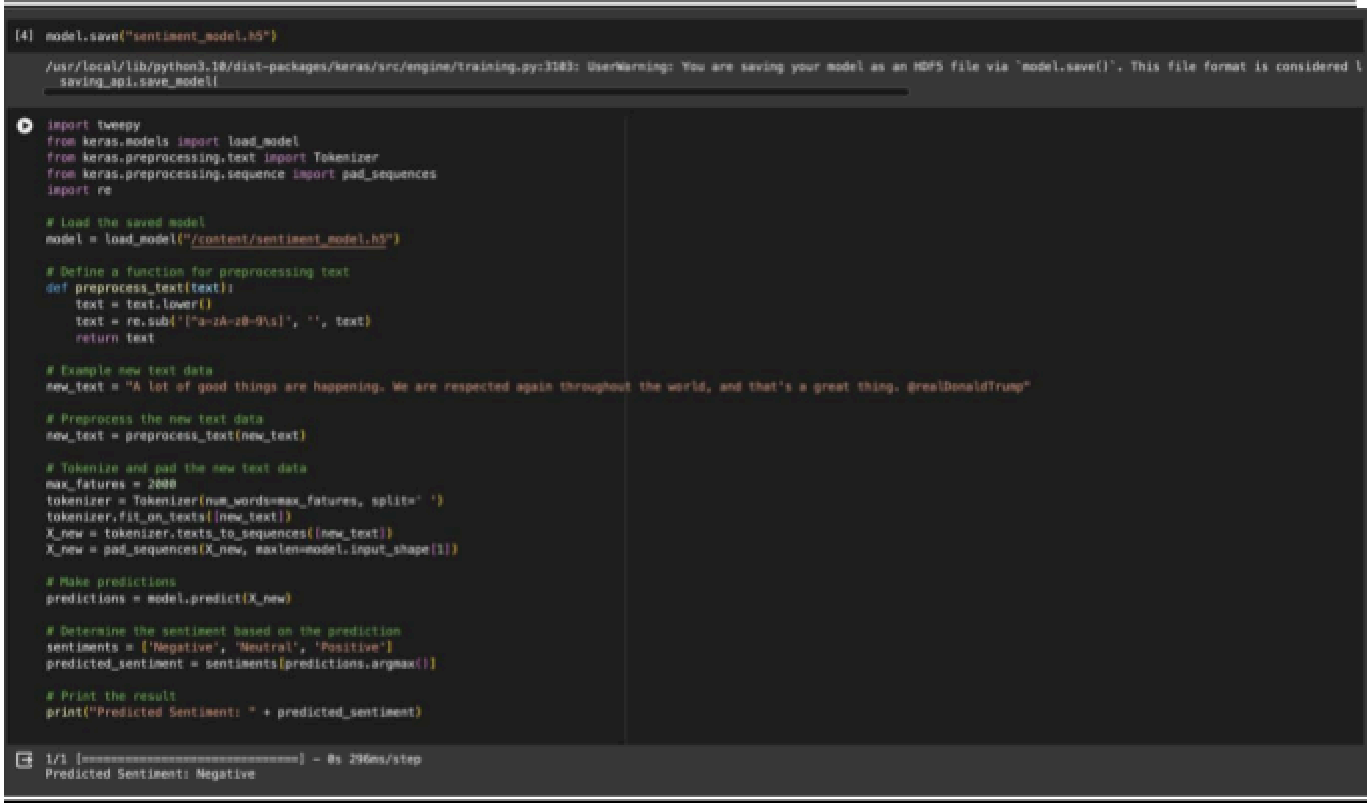
Q1:
Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump”)

Code:



```
from tensorflow.keras.utils import to_categorical
import re

from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('/content/Sentiment (3).csv')
# Keeping only the neccessary columns
data = data[['text','sentiment']]

data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)

X = pad_sequences(X)

embed_dim = 128
lstm_out = 196
def createmodel():
    model = Sequential()
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
    return model
# print(model.summary())

labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33, random_state = 42)

batch_size = 32
model = createmodel()
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2)
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
print(score)
print(acc)
print(model.metrics_names)
```

```
291/291 - 48s - loss: 0.8208 - accuracy: 0.6428 - 48s/epoch - 166ms/step
144/144 - 4s - loss: 0.7660 - accuracy: 0.6614 - 4s/epoch - 31ms/step
0.7660231725692749
0.6614242196083069
['loss', 'accuracy']
```



```
[4] model.save("sentiment_model.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered l
  saving_api.save_model(
```

```
import tweepy
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import re

# Load the saved model
model = load_model("/content/sentiment_model.h5")

# Define a function for preprocessing text
def preprocess_text(text):
    text = text.lower()
    text = re.sub('[^a-zA-z0-9\s]', '', text)
    return text

# Example new text data
new_text = "A lot of good things are happening. We are respected again throughout the world, and that's a great thing. @realDonaldTrump"

# Preprocess the new text data
new_text = preprocess_text(new_text)

# Tokenize and pad the new text data
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts([new_text])
X_new = tokenizer.texts_to_sequences([new_text])
X_new = pad_sequences(X_new, maxlen=model.input_shape[1])

# Make predictions
predictions = model.predict(X_new)

# Determine the sentiment based on the prediction
sentiments = ['Negative', 'Neutral', 'Positive']
predicted_sentiment = sentiments[predictions.argmax()]

# Print the result
print("Predicted Sentiment: " + predicted_sentiment)
```

```
1/1 [==============================] - 0s 296ms/step
Predicted Sentiment: Negative
```

Explanation:

1. Import Libraries:It starts by importing necessary libraries. `tweepy` is used for accessing the Twitter API, `keras` is used for building and loading the neural network model, and `re` for regular expression operations.

2. Load Pre-trained Model:The pre-trained sentiment analysis model is loaded from a saved file (`sentiment_model.h5`). This model is assumed to be trained to classify text into sentiments.

3. Preprocess Text:The `preprocess_text` function is defined to clean the input text by converting it to lowercase and removing non-alphanumeric characters. This ensures the model receives the text in the format it expects.

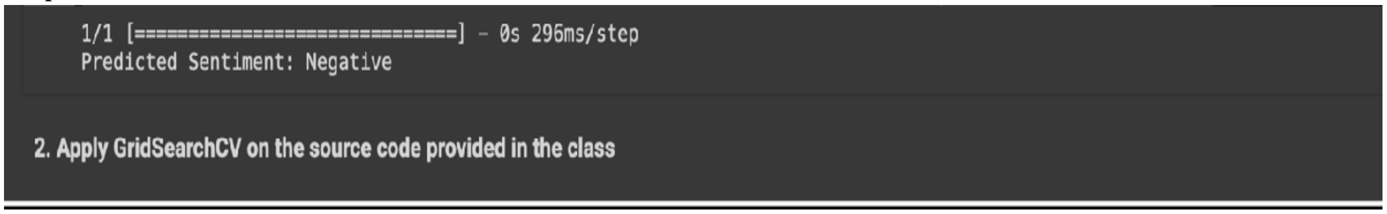
4. Example Text:A sample tweet is provided as `new_text`. This text is then preprocessed to remove unwanted characters and format it properly.

5. Tokenize and Pad the Text: The text is tokenized using Keras' `Tokenizer`, which converts the text into a sequence of integers where each integer represents a specific word in a dictionary. The sequence is then padded to ensure it has a fixed length, matching the model's input requirements.

6. Make Predictions:The preprocessed and formatted text is fed into the model to predict its sentiment. The model outputs a probability distribution across the possible sentiment classes (Negative, Neutral, Positive).

7. Determine Sentiment: The sentiment with the highest probability is selected as the predicted sentiment for the input text.

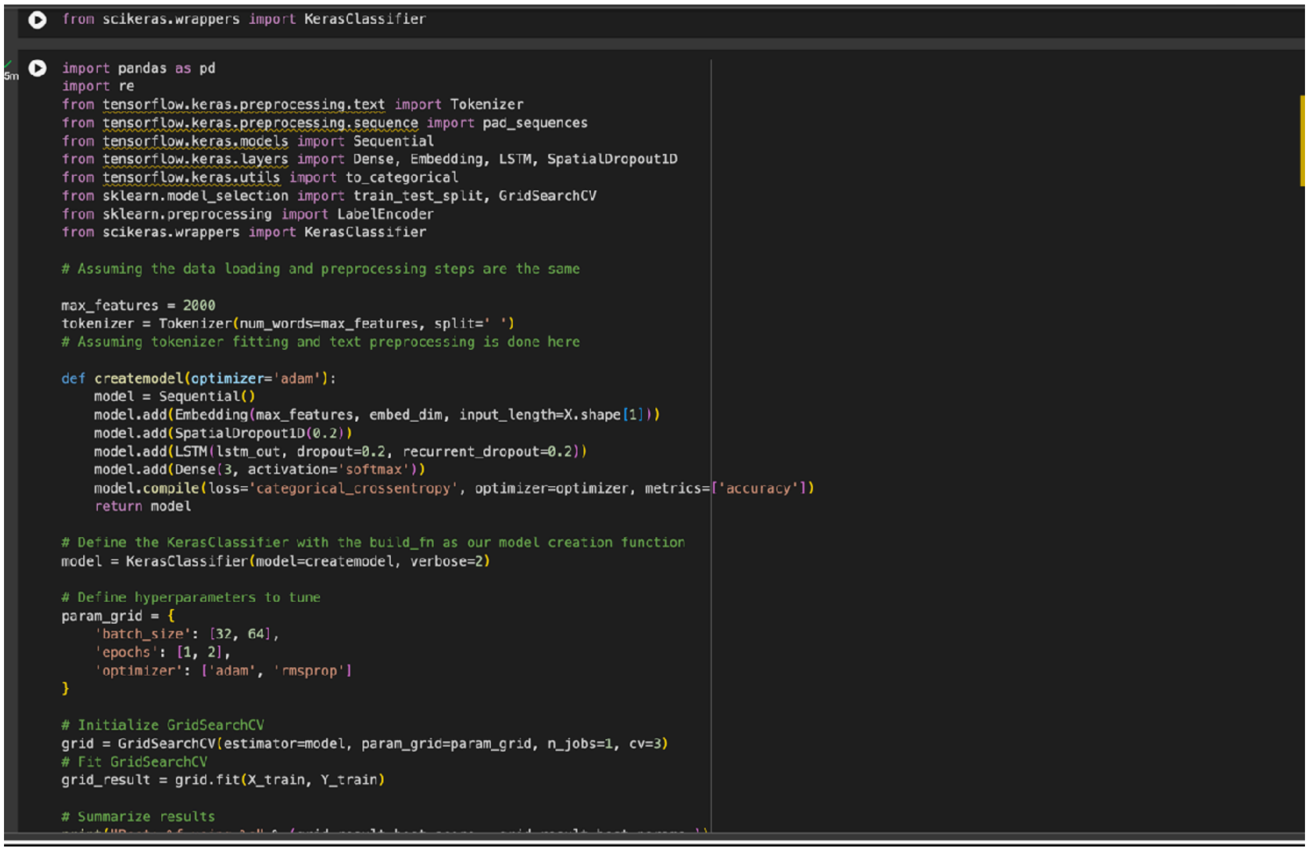
Output:



```
1/1 [==============================] - 0s 296ms/step
Predicted Sentiment: Negative
```

2. Apply GridSearchCV on the source code provided in the class

Q2:
Code:



```
from scikeras.wrappers import KerasClassifier
```

```
import pandas as pd
import re
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from scikeras.wrappers import KerasClassifier

# Assuming the data loading and preprocessing steps are the same

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
# Assuming tokenizer fitting and text preprocessing is done here

def createmodel(optimizer='adam'):
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

# Define the KerasClassifier with the build_fn as our model creation function
model = KerasClassifier(model=createmodel, verbose=2)

# Define hyperparameters to tune
param_grid = {
    'batch_size': [32, 64],
    'epochs': [1, 2],
    'optimizer': ['adam', 'rmsprop']
}

# Initialize GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)
# Fit GridSearchCV
grid_result = grid.fit(X_train, Y_train)

# Summarize results
```

Explanation:

1. Library Imports: It starts by importing necessary libraries. `pandas` for data manipulation, `re` for regular expressions, `tensorflow.keras` for building and training the neural network model, `sklearn.model_selection` for splitting the dataset and conducting grid search, and `scikeras.wrappers` to wrap Keras models for use with scikit- learn.

2. Model Building Function:The `createmodel` function defines the architecture of the neural network using Keras' Sequential API. It includes an Embedding layer for text input, a SpatialDropout1D layer to reduce overfitting, an LSTM layer for learning from the sequence data, and a Dense output layer with a softmax activation function for classification. The optimizer for compiling the model can be adjusted, making the model flexible for hyperparameter tuning.

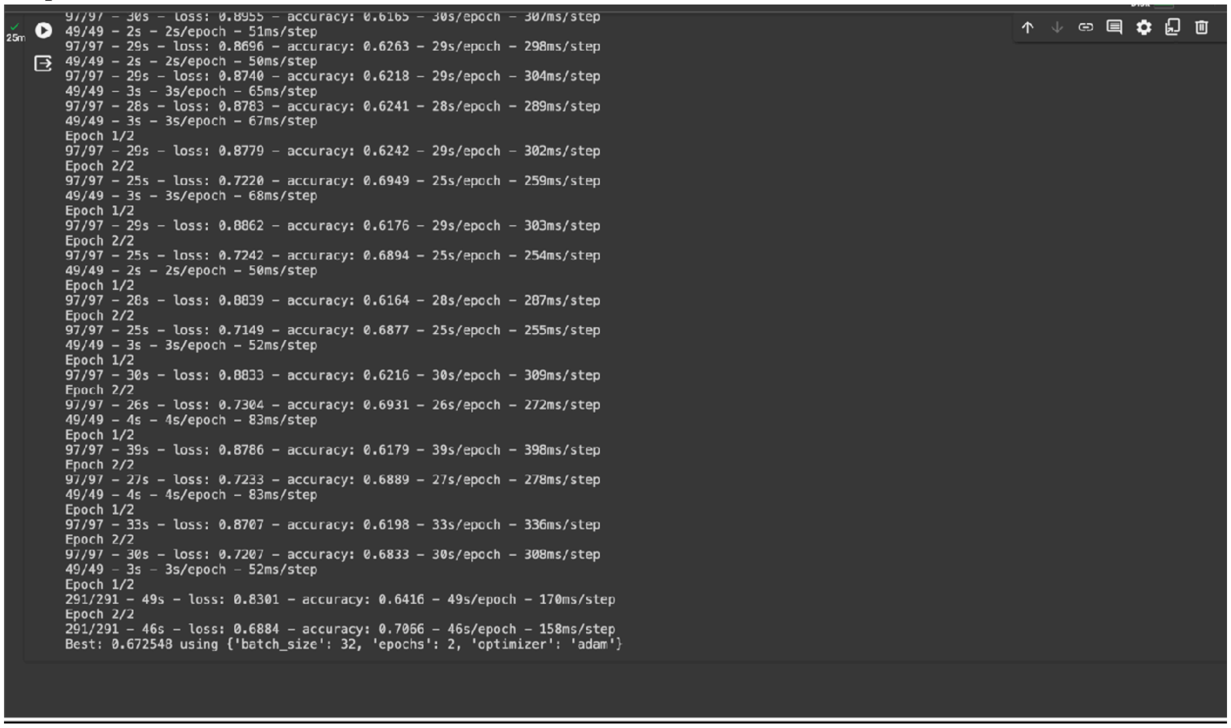
3. KerasClassifier Wrapper: A `KerasClassifier` wrapper is used to make the Keras model compatible with scikit- learn's grid search functionality. This allows the use of scikit-learn's `GridSearchCV` for hyperparameter tuning.

4. Hyperparameter Tuning:A parameter grid is defined with different values for batch size, number of epochs, and optimizer type. `GridSearchCV` is then used to exhaustively search through the parameter grid for the best model configuration based on cross-validation performance. It evaluates model performance for each combination of parameters across a specified number of folds of the training data.

5. Model Training and Selection: `grid.fit(X_train, Y_train)` trains the model using the training data across all combinations of parameters specified in `param_grid`, using cross-validation. After fitting, it identifies the combination of parameters that resulted in the best model performance.

6. Results Summary: Finally, the best performance score and the hyperparameters that led to this best score are printed. This provides insights into which settings worked best for the given text classification task.

Output:



```
97/97 - 30s - loss: 0.8955 - accuracy: 0.6165 - 30s/epoch - 307ms/step
49/49 - 2s - 2s/epoch - 51ms/step
97/97 - 29s - loss: 0.8696 - accuracy: 0.6263 - 29s/epoch - 298ms/step
49/49 - 2s - 2s/epoch - 50ms/step
97/97 - 29s - loss: 0.8740 - accuracy: 0.6218 - 29s/epoch - 304ms/step
49/49 - 3s - 3s/epoch - 65ms/step
97/97 - 28s - loss: 0.8783 - accuracy: 0.6241 - 28s/epoch - 289ms/step
49/49 - 3s - 3s/epoch - 67ms/step
Epoch 1/2
97/97 - 29s - loss: 0.8779 - accuracy: 0.6242 - 29s/epoch - 302ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7220 - accuracy: 0.6949 - 25s/epoch - 259ms/step
49/49 - 3s - 3s/epoch - 68ms/step
Epoch 1/2
97/97 - 29s - loss: 0.8862 - accuracy: 0.6176 - 29s/epoch - 303ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7242 - accuracy: 0.6894 - 25s/epoch - 254ms/step
49/49 - 2s - 2s/epoch - 50ms/step
Epoch 1/2
97/97 - 28s - loss: 0.8839 - accuracy: 0.6164 - 28s/epoch - 287ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7149 - accuracy: 0.6877 - 25s/epoch - 255ms/step
49/49 - 3s - 3s/epoch - 52ms/step
Epoch 1/2
97/97 - 30s - loss: 0.8833 - accuracy: 0.6216 - 30s/epoch - 309ms/step
Epoch 2/2
97/97 - 26s - loss: 0.7304 - accuracy: 0.6931 - 26s/epoch - 272ms/step
49/49 - 4s - 4s/epoch - 83ms/step
Epoch 1/2
97/97 - 39s - loss: 0.8786 - accuracy: 0.6179 - 39s/epoch - 398ms/step
Epoch 2/2
97/97 - 27s - loss: 0.7233 - accuracy: 0.6889 - 27s/epoch - 278ms/step
49/49 - 4s - 4s/epoch - 83ms/step
Epoch 1/2
97/97 - 33s - loss: 0.8707 - accuracy: 0.6198 - 33s/epoch - 336ms/step
Epoch 2/2
97/97 - 30s - loss: 0.7207 - accuracy: 0.6833 - 30s/epoch - 308ms/step
49/49 - 3s - 3s/epoch - 52ms/step
Epoch 1/2
291/291 - 49s - loss: 0.8301 - accuracy: 0.6416 - 49s/epoch - 170ms/step
Epoch 2/2
291/291 - 46s - loss: 0.6884 - accuracy: 0.7066 - 46s/epoch - 158ms/step
Best: 0.672548 using {'batch_size': 32, 'epochs': 2, 'optimizer': 'adam'}
```