- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
import numpy as np
def load_data():
 # Load motivational quotes and demotivational quotes from files
    with open('happy quotes.txt', 'r', encoding='utf-8') as f:
        happy_quotes = f.readlines()
    with open('sad quotes.txt', 'r', encoding='utf-8') as f:
        sad_quotes = f.readlines()
 # Combine both classes of quotes and create labels (1 for motivational, 0 for demotivational)
    quotes = happy_quotes + sad_quotes
    labels = np.concatenate([np.ones(len(happy_quotes)), np.zeros(len(sad_quotes))])
    return quotes, labels
quotes, labels = load_data()
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

def preprocess_data(quotes):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(quotes)
    sequences = tokenizer.texts_to_sequences(quotes)
    vocab_size = len(tokenizer.word_index) + 1

    # Pad the sequences to have the same length
    max_sequence_length = max(len(seq) for seq in sequences)
    padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post')

    return padded_sequences, vocab_size

X, vocab_size = preprocess_data(quotes)
```

```
load_data()
```

```
     (['1.ஒவ்வொரு சிறிய மாற்றமும் பெரிய வெற்றியின் ஒரு பகுதியாகும்\n',
       '2.நம்பிக்கை வெற்றியோடு வரும். ஆனால் வெற்றி நம்பிக்கை உள்ளோரிடம் மட்டுமே வரும்.\n',
       '3.மனம் உங்களைக் கட்டுப்படுத்தும் முன் உங்கள் மனதைக் கட்டுப்படுத்துங்கள்\n',
       '4.நீங்கள் நிறுத்தாத வரை எவ்வளவு மெதுவாகச் சென்றாலும் பரவாயில்லை.\n',
       '5.தைரியம் பயத்தை விட ஒரு படி மேலே உள்ளது.\n',
       '6.அறிவை விட முக்கியமானது, உங்கள் இலக்கை அடைய உங்கள் விருப்பம்.\n',
       '7.செய்ய முடிந்தவன் சாதிக்கிறான், செய்ய முடியாதவன் போதிக்கிறான்.\n',
       '8.மலையைப் பார்த்து மலைத்து விடாதே, மலை மீது ஏறினால் அதுவும் உன் காலடியில்.\n',
       '9.முயலும் வெல்லும், ஆமையும் வெல்லும், ஆனால் முயலாமை என்றுமே வெல்லாது.\n',
       '10.இன்று நீங்கள் உணரும் வலி நாளை நீங்கள் உணரும் பலமாக இருக்கும்.\n',
       '11.நாம் வலியைத் தழுவி, அதை நமது பயணத்திற்கு எரிபொருளாக எரிக்க வேண்டும்.\n',
       '12.வெற்றி இலக்கை அடைய தோல்விகள் படிகட்டுகள்.\n',
       '13.ஒரு சிக்கல் உங்கள் சிறந்ததைச் செய்வதற்கான வாய்ப்பாகும்.\n',
       '14.உங்களால் பறக்க முடியாவிட்டால் ஓடுங்கள்.\n',
       '15.பழைய பழக்கங்கள் புதிய பாதைகளைத் திறக்காது.\n',
       '16.நம்மீது நம்பிக்கை நமக்கிருக்கும் வரை வாழ்க்கை நம்வசம்.\n',
       '17.விழுவதெல்லாம் எழுவதற்குத்தானே தவிர அழுவதற்கு அல்ல.\n',
       '18.நீ இன்று செய்யும் சின்ன சின்ன முயற்சிகள் நாளை மாறும் வெற்றியின் ஆணி வேர்கள்.\n',
       '19.உன் திறமையை வெளி காட்டு, உலகம் உன்னை கண்டறியும்.\n',
       '20சிந்தனை மட்டும் செய்ய உனக்கு தெரியுமானால் நீயே உனக்கான மிகச்சிறந்த ஆலோசகர்.\n',
       '1.முகம் மனதை தேடுகிறது, ஆனால் மனம் முகத்தை காணலாம்.\n',
       '2.மன நோய் என்பது மனதால் வருவதல்ல சில மனிதர்களால் வருகிறது…\n',
       '3.முடிந்த காரியத்தின் பின் ஏன் மீண்டும் அதற்கு விண்ணப்பிக்கிறீர்கள்?\n',
       '4.நீ உன் மனதை நிறைவேற்றினால், முகம் தேய்ப்படும்.\n',
       '5.நல்லவர் பயன்படுத்தாத முட்டாள்களின் ஆயுதம் "கோபம்\n',
       ' 6.கோபம் என்னும் இருளில் மூழ்கியவர்கு, பாசம் என்னும் விடிவெள்ளி தெரிவதில்லை\n',
       ' 7.கோபம்: காட்டப்படும் இடத்தை விட இருக்கும் இடத்தையே அதிகம் பாதிக்கும். \n',
       '8.நல்லவர் பயன்படுத்தாத முட்டாள்களின் ஆயுதம் "கோபம்\n',
       '9.கோபம்" சிலருக்கு "பிறவி குணம்". சிலருக்கு அது "பிறரால் பிறக்கும் குணம்"\n',
       '10.கோபம் என்னும் இருளில் மூழ்கியவர்கு, பாசம் என்னும் விடிவெள்ளி தெரிவதில்லை!\xa0\n',
       '11.\xa0கோபம்: காட்டப்படும் இடத்தை விட இருக்கும் இடத்தையே அதிகம் பாதிக்கும்.\n',
       '12.உன்னை கைவிடுவதற்கு நான் ஏற்றுக்கொள்கிறேன், நன்றி சொல்ல கூடாது.\n',
       '13.தூங்க விட்டு நீ விரும்பினாயே நான் விரும்பத்தக்கது என்று அனுப்புகின்றேன்\n',
       '14.செல்வி என்ற தீயத்தை மட்டுமே வெல்வது நலமாக உள்ளது.\n',
       '15.என் முகத்தில் உண்மை இல்லை, ஆனால் உன் முகத்தில் எனக்கு மிகுந்த நாசம் உள்ளது.\n',
       '16.பெரிய எலி உன்னை விழுத்துக்கொள்வது எப்படி என்று தெரியுமா?\n',
       '17.குழந்தைகளுக்கு வாய்த்த மருந்துகள் பெரும்பாலும் பொருந்தாது.\n',
       '18.எப்படி இந்த பொய்யும் திரும்பி வருகின்றீர்கள் என்று அனுப்பத்தக்கது என்ன?\n',
       '19.உனக்கு எதுவும் பெரிய வலிமை இல்லை என்று தெரியுமா?\n',
```

```
       '20.நீங்கள் செய்தது போல் நான் உங்களைப் புறக்கணிக்க விரும்புகிறேன்.\n',
       '\n'],
      array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
             1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0.]))
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

def create_model(nodes=32, layers=1):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=nodes, input_length=X.shape[1]))

    for _ in range(layers):
        model.add(LSTM(nodes, return_sequences=True))
    model.add(LSTM(nodes))

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

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Double-click (or enter) to edit

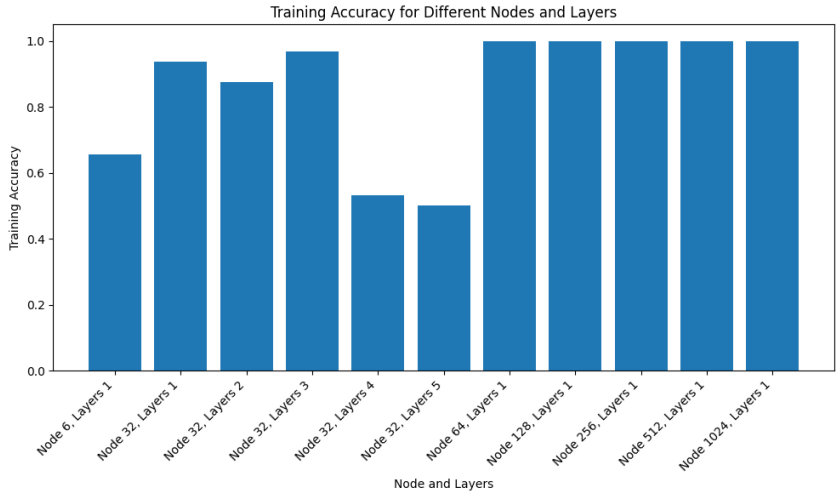
```
from sklearn.model_selection import train_test_split
import time
import matplotlib.pyplot as plt

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

# Function to train and evaluate the model
def train_and_evaluate_model(nodes, layers):
    model = create_model(nodes=nodes, layers=layers)
    start_time = time.time()
    model.fit(X_train, y_train, epochs=5, batch_size=128, verbose=1)
    end_time = time.time()
    _, train_accuracy = model.evaluate(X_train, y_train)
    _, test_accuracy = model.evaluate(X_test, y_test)
    return train_accuracy, test_accuracy, end_time - start_time

# Define the configurations to evaluate
nodes_list = [6, 32, 64, 128, 256, 512, 1024]
layers_list = [1, 2, 3, 4, 5]

# Create a dictionary to store the results for each configuration
results = {}

# Find the best configuration
best_accuracy = 0
best_config = None

# Evaluate each configuration and store the components
for nodes in nodes_list:
    if nodes == 32:  # Evaluate all layers for node 32
        for layers in layers_list:
            train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
            results[(nodes, layers)] = {
                "Train Accuracy": train_accuracy,
                "Test Accuracy": test_accuracy,
                "Running Time": running_time
            }
            if test_accuracy > best_accuracy:
                best_accuracy = test_accuracy
                best_config = (nodes, layers)
    else:  # Only evaluate the first layer for other nodes
        layers = 1
        train_accuracy, test_accuracy, running_time = train_and_evaluate_model(nodes, layers)
        results[(nodes, layers)] = {
            "Train Accuracy": train_accuracy,
            "Test Accuracy": test_accuracy,
            "Running Time": running_time
        }
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_config = (nodes, layers)

# Print the results for all configurations
for config, values in results.items():
    node, layers = config
```

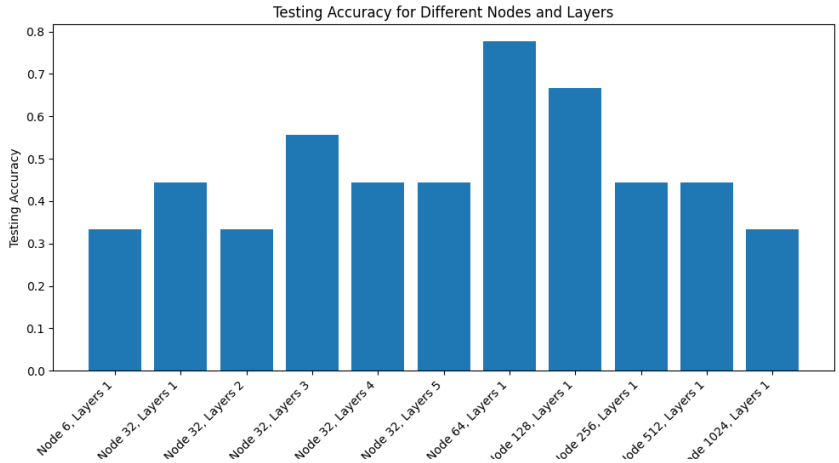
```
    print("Node: {}, Layers: {}".format(node, layers))
    print("Training Accuracy:", values["Train Accuracy"])
    print("Testing Accuracy:", values["Test Accuracy"])
    print("Running Time:", values["Running Time"])
    print("-" * 30)
```

```
:====================] - 4s 4s/step - loss: 0.6935 - accuracy: 0.5000

:====================] - 0s 18ms/step - loss: 0.6932 - accuracy: 0.5312

:====================] - 0s 17ms/step - loss: 0.6928 - accuracy: 0.5938

:====================] - 0s 18ms/step - loss: 0.6924 - accuracy: 0.5938

:====================] - 0s 16ms/step - loss: 0.6920 - accuracy: 0.6875
:====================] - 1s 848ms/step - loss: 0.6916 - accuracy: 0.6562
:====================] - 1s 832ms/step - loss: 0.6950 - accuracy: 0.3333

:====================] - 5s 5s/step - loss: 0.6932 - accuracy: 0.4688

:====================] - 0s 23ms/step - loss: 0.6925 - accuracy: 0.5000

:====================] - 0s 20ms/step - loss: 0.6917 - accuracy: 0.6250

:====================] - 0s 21ms/step - loss: 0.6907 - accuracy: 0.8125

:====================] - 0s 18ms/step - loss: 0.6896 - accuracy: 0.9062
:====================] - 1s 823ms/step - loss: 0.6881 - accuracy: 0.9375
:====================] - 1s 1s/step - loss: 0.6938 - accuracy: 0.4444

:====================] - 6s 6s/step - loss: 0.6931 - accuracy: 0.4688

:====================] - 0s 23ms/step - loss: 0.6928 - accuracy: 0.5000

:====================] - 0s 39ms/step - loss: 0.6924 - accuracy: 0.7188

:====================] - 0s 46ms/step - loss: 0.6919 - accuracy: 0.5625

:====================] - 0s 39ms/step - loss: 0.6912 - accuracy: 0.6875
'flow:5 out of the last 5 calls to <function Model.make_test_function.<locals>.test_function at 0x7800c1a0f880> triggered tf.functi
:====================] - 2s 2s/step - loss: 0.6903 - accuracy: 0.8750
'flow:6 out of the last 6 calls to <function Model.make_test_function.<locals>.test_function at 0x7800c1a0f880> triggered tf.functi
:====================] - 1s 1s/step - loss: 0.6943 - accuracy: 0.3333

:====================] - 8s 8s/step - loss: 0.6932 - accuracy: 0.4375

:====================] - 0s 30ms/step - loss: 0.6929 - accuracy: 0.5000

:====================] - 0s 28ms/step - loss: 0.6926 - accuracy: 0.5312

:====================] - 0s 27ms/step - loss: 0.6921 - accuracy: 0.5000

:====================] - 0s 29ms/step - loss: 0.6915 - accuracy: 0.7500
:====================] - 2s 2s/step - loss: 0.6907 - accuracy: 0.9688
:====================] - 2s 2s/step - loss: 0.6927 - accuracy: 0.5556

:====================] - 14s 14s/step - loss: 0.6931 - accuracy: 0.6250

:====================] - 0s 73ms/step - loss: 0.6929 - accuracy: 0.5000

:====================] - 0s 69ms/step - loss: 0.6927 - accuracy: 0.5000
```

```
# Plot the graphs for Training Accuracy, Testing Accuracy, and Running Time for all nodes with their layers
x_labels = ["Node {}, Layers {}".format(node, layers) for node, layers in results.keys()]
train_accuracies = [values["Train Accuracy"] for values in results.values()]
test_accuracies = [values["Test Accuracy"] for values in results.values()]
running_times = [values["Running Time"] for values in results.values()]

# Plot the bar graph for Training Accuracy
plt.figure(figsize=(10, 6))
plt.bar(x_labels, train_accuracies)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Training Accuracy")
plt.title("Training Accuracy for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Plot the bar graph for Testing Accuracy
plt.figure(figsize=(10, 6))
plt.bar(x_labels, test_accuracies)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Testing Accuracy")
```

```
plt.title("Testing Accuracy for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Plot the bar graph for Running Time
plt.figure(figsize=(10, 6))
plt.bar(x_labels, running_times)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Node and Layers")
plt.ylabel("Running Time (seconds)")
plt.title("Running Time for Different Nodes and Layers")
plt.tight_layout()
plt.show()

# Print the results for the best configuration
print("BEST CONFIGURATION:", best_config)
print("Components of the Best Configuration:")
print("Training Accuracy:", results[best_config]["Train Accuracy"])
print("Testing Accuracy:", results[best_config]["Test Accuracy"])
print("Running Time:", results[best_config]["Running Time"])
```






```
# Function to read text from a file and return the words
def read_text_from_file(file_path):
    with open(file_path, "r", encoding="utf-8") as file:
        text = file.read()
    return text

# Function to clean and tokenize the text into words
def preprocess_text(text):
    # Remove unwanted characters and split the text into words
    words = text.strip().replace("\n", " ").split(" ")
    # Remove any empty strings from the list of words
    words = [word.strip() for word in words if word.strip()]
    return words

# Function to count the word frequency in a list of words
def count_word_frequency(words):
    word_frequency = {}
    for word in words:
        if word in word_frequency:
            word_frequency[word] += 1
        else:
            word_frequency[word] = 1
    return word_frequency

# Read and preprocess the motivational_quotes.txt
motivational_quotes_text = read_text_from_file("happy quotes.txt")
motivational_quotes_words = preprocess_text(motivational_quotes_text)
motivational_quotes_word_frequency = count_word_frequency(motivational_quotes_words)

# Read and preprocess the demotivational_quotes.txt
demotivational_quotes_text = read_text_from_file("sad quotes.txt")
demotivational_quotes_words = preprocess_text(demotivational_quotes_text)
demotivational_quotes_word_frequency = count_word_frequency(demotivational_quotes_words)

# Display the word frequency for motivational_quotes
print("Word Frequency for motivational_quotes:")
for word, frequency in motivational_quotes_word_frequency.items():
    print(f"{word}: {frequency}")

# Display the word frequency for demotivational_quotes
print("\nWord Frequency for demotivational_quotes:")
for word, frequency in demotivational_quotes_word_frequency.items():
    print(f"{word}: {frequency}")
```

```
Word Frequency for motivational_quotes:
1.ஒவ்வொரு: 1
சிறிய: 1
மாற்றமும்: 1
பெரிய: 1
வெற்றியின்: 2
ஒரு: 2
பகுதியாகும்: 1
2.நம்பிக்கை: 1
வெற்றியோடு: 1
வரும்.: 2
ஆனால்: 2
வெற்றி: 1
நம்பிக்கை: 2
உள்ளோரிடம்: 1
மட்டுமே: 1
3.மனம்: 1
உங்களைக்: 1
கட்டுப்படுத்தும்: 1
முன்: 1
உங்கள்: 4
மனதைக்: 1
கட்டுப்படுத்துங்கள்: 1
4.நீங்கள்: 1
நிறுத்தாத: 1
வரை: 2
எவ்வளவு: 1
மெதுவாகச்: 1
சென்றாலும்: 1
பரவாயில்லை.: 1
5.தைரியம்: 1
பயத்தை: 1
விட: 2
படி: 1
மேலே: 1
உள்ளது.: 1
6.அறிவை: 1
முக்கியமானது,: 1
இலக்கை: 2
அடைய: 2
விருப்பம்.: 1
7.செய்ய: 1
முடிந்தவன்: 1
சாதிக்கிறான்,: 1
செய்ய: 2
முடியாதவன்: 1
போதிக்கிறான்.: 1
8.மலையைப்: 1
பார்த்து: 1
மலைத்து: 1
விடாதே,: 1
மலை: 1
மீது: 1
ஏறினால்: 1
அதுவும்: 1
உன்: 1
காலடியில்.: 1
9.முயலும்: 1
```

```
!pip install prettytable
```

```
Requirement already satisfied: prettytable in /usr/local/lib/python3.10/dist-packages (0.7.2)
```

```
!pip install tabulate
```

```
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (0.9.0)
```

```
from tabulate import tabulate

# Function to read text from a file and return the words
def read_text_from_file(file_path):
    with open(file_path, "r", encoding="utf-8") as file:
        text = file.read()
    return text

# Function to clean and tokenize the text into words
def preprocess_text(text):
    # Remove unwanted characters and split the text into words
    words = text.strip().replace("\n", " ").split(" ")
    # Remove any empty strings from the list of words
    words = [word.strip() for word in words if word.strip()]
    return words

# Function to count the word frequency in a list of words
def count_word_frequency(words):
    word_frequency = {}
```