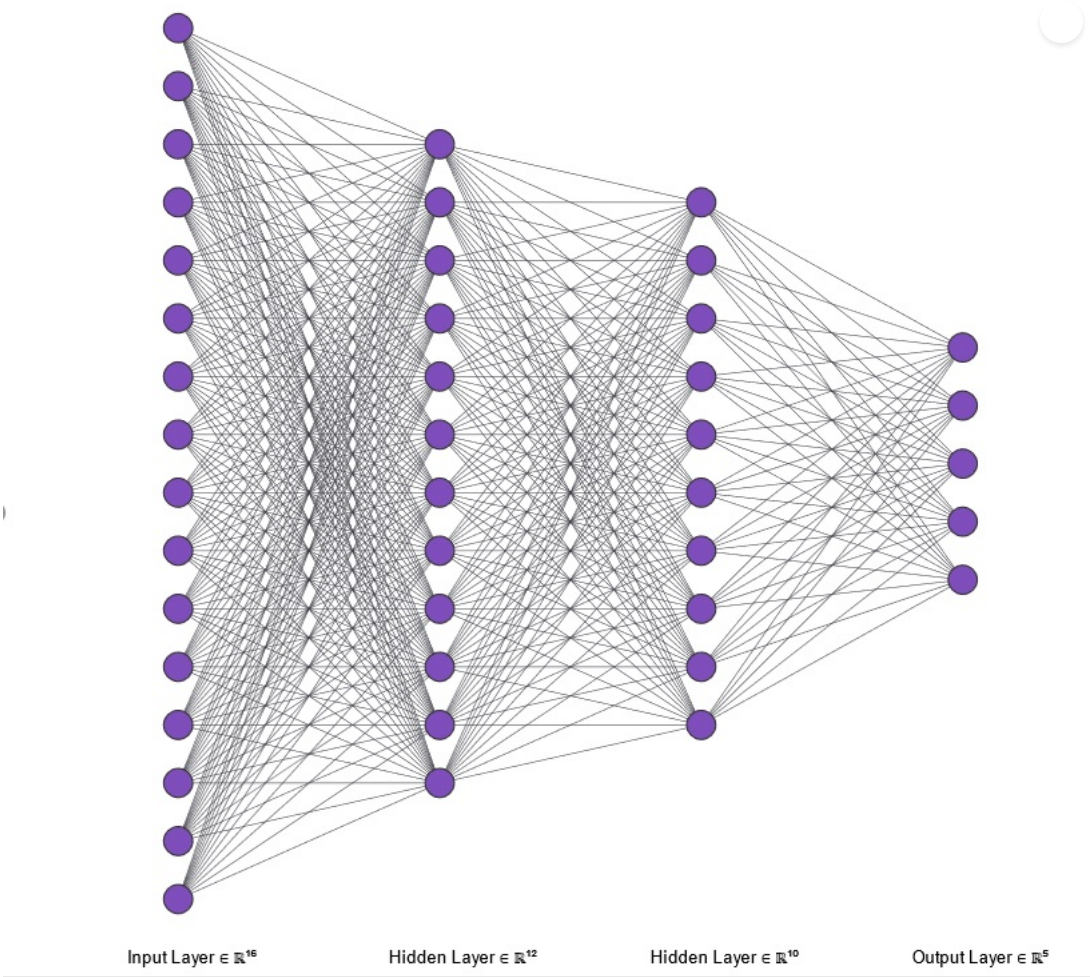
Sequenzvorhersage

Künstliche Neuronale Netz (ANN)

Dabei handelt es sich um einen Deep-Learning-Algorithmus, der Informationen auf ähnliche Weise wie das menschliche Gehirn verarbeitet. Es wird häufig verwendet, um das Muster zwischen den Eingabemerkmalen und der entsprechenden Ausgabe in einem Datensatz zu erstellen und zu lernen. Die nächste Abbildung zeigt die Architektur des neuronalen Netzwerks:

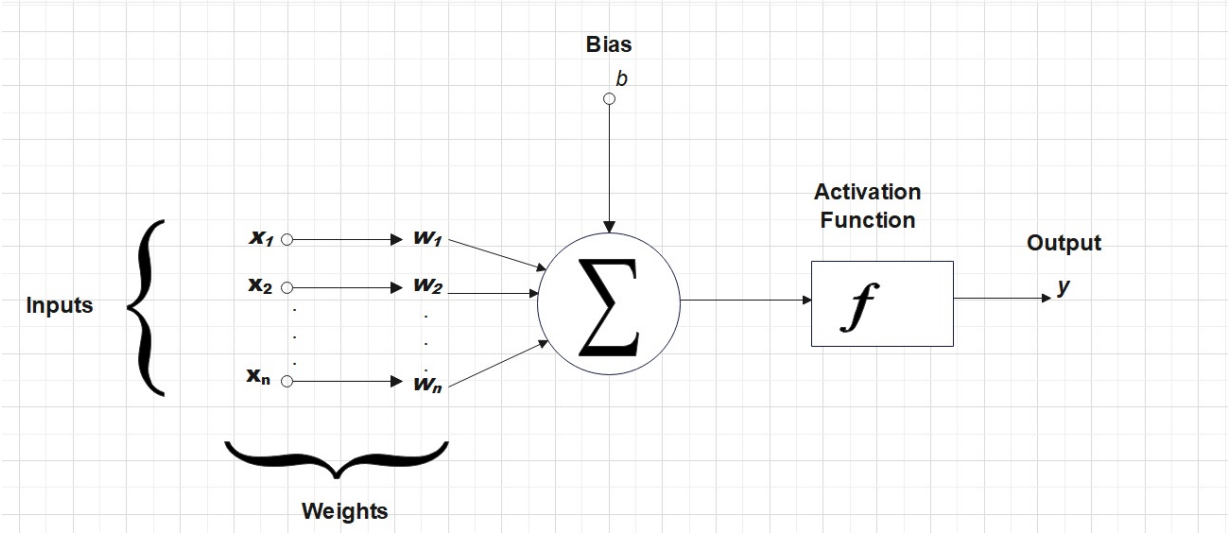


Die erste Schicht besteht aus einer Reihe von Neuronen. Diese Ebene sollte die gleiche Größe wie der Eingabevektor xi haben, wobei i = 1, 2, ….., D, der die Merkmale des Datensatzes darstellt. Die verborgenen Schichten haben die Aufgabe, mathematische Berechnungen an der Eingabe durchzuführen, um die Ausgabe zu erzeugen, die an die Ausgabeschicht weitergeleitet werden soll. In einem neuronalen Klassifizierungsnetzwerk sollte die Dimension der Ausgabeschicht der Anzahl der Klassen oder Beschriftungen im Datensatz in einem neuronalen Klassifizierungsnetzwerk entsprechen. Beispielsweise erfordert ein Datensatz mit 4 Merkmalen und 3 Klassen ein neuronales Netzwerk mit 4 Eingabeneuronen und 3 Ausgabeneuronen, um das Eingabe- und Ausgabemuster zu lernen.
Der Algorithmus des neuronalen Netzwerks umfasst zwei Schritte:

1. Vorwärtsdurchlauf von Daten durch ein Feed-Forward-Neuronales Netzwerk.
2. Rückausbreitung von Fehlern zum Trainieren eines neuronalen Netzwerks.

1. Vorwärtsdurchlauf durch ein Feed-Forward-Neuronales Netzwerk:

Bei dieser Methode werden die Informationen über die verborgenen Schichten an die Ausgabeschicht des Netzwerks weitergeleitet. Die Neuronen der ersten verborgenen Schicht erhalten eine gewichtete Summe von Werten des Eingabevektors zusammen mit einem Bias-Term b, wie in Abb. 2 dargestellt. Anschließend wendet jedes Neuron eine differenzierbare, nichtlineare Aktivierung auf diese gewichtete Summe an, um einen Ausgabewert zu ergeben . Diese Schritte sind in der folgenden Abbildung dargestellt:



Ebenso verwenden Neuronen nachfolgender Schichten eine gewichtete Summe der Ausgaben aller Neuronen der vorherigen Schicht zusammen mit einem Bias-Term als Eingabe. Dieser Prozess wird fortgesetzt, bis die Neuronen der Ausgabeschicht ihre Ausgabewahrscheinlichkeitswerte der Klassen bereitstellen. Dem Eingabevektor wird eine Klasse zugewiesen, deren entsprechendes Neuron den höchsten Wahrscheinlichkeitswert hat.

In einem neuronalen Netzwerk verfügt jede Schicht über eine eigene Aktivierungsfunktion. Die für ChatBot verwendeten Aktivierungsfunktionen sind beispielsweise die Rectified Linear Unit (ReLu)-Funktion für verborgene Schichten und die Softmax-Funktion für die Ausgabeschicht. Die ReLu-Funktion ist definiert als:

$$f(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}$$

Und die Softmax-Funktion ist definiert als:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Wobei Vektor Z der Eingabevektor, e^{z_i} Standard-Exponentialfunktion für Eingabevektor, K die Anzahl der Klassen im Mehrklassenklassifikator, e^{z_j} Standard-Exponentialfunktion für Ausgabevektor der Klasse j.

2. Fehlerrückausbreitung zum Trainieren eines neuronalen Netzwerks:

Dieser Schritt ist der wichtigste, da die Hauptaufgabe des Algorithmus des neuronalen Netzwerks darin besteht, den richtigen Satz von Gewichten für alle Schichten zu lernen, die zur richtigen Ausgabe führen. Dies beinhaltet die Weitergabe des am Ausgang berechneten Fehlers oder Verlusts nach hinten, sodass die neuen Gewichte und Verzerrungen gelernt werden.

In neuronalen Netzen werden je nach Modelltyp zwei Haupttypen von Verlustfunktionen verwendet:

1. Regressionsverlustfunktionen: Mittlerer quadratischer Fehler, mittlerer absoluter Fehler. Es werden in der Regression neuronale Netze eingesetzt.
2. Klassifizierungsverlustfunktionen: Binäre Kreuzentropie, kategoriale Kreuzentropie. Diese Art von Verlustfunktion wird für Klassifizierungsaufgaben verwendet.

Im Folgenden sind die Formeln der MSE- und Kreuzentropieverlustfunktionen aufgeführt:

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n}(\hat{y}_i - y_i)^2$$

MSE ist der Durchschnitt der quadrierten Differenzen zwischen den tatsächlichen und den vorhergesagten Werten.

Für die Klassifizierung mehrerer Klassen, bei der die Anzahl der Klassen M > 2 beträgt, wird die kategoriale Kreuzentropie verwendet:

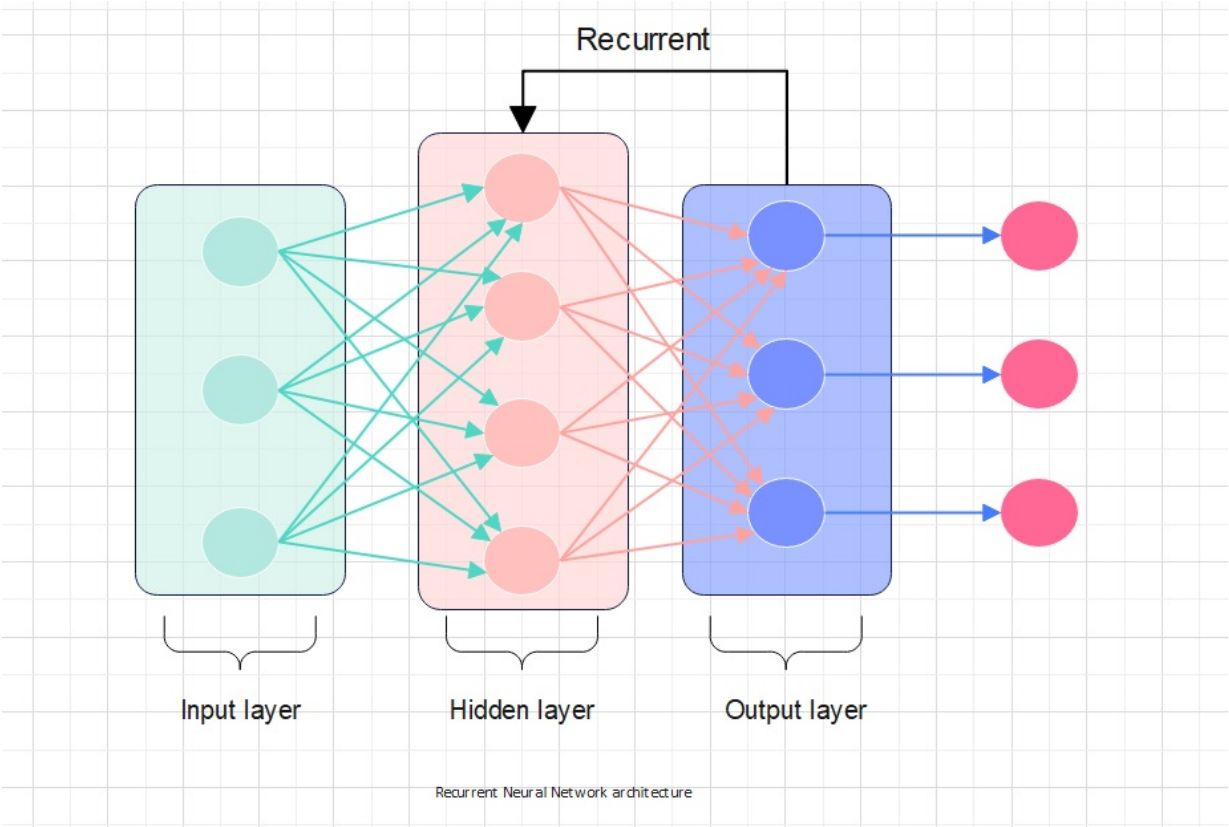
$$\text{CE loss} = -\sum_{c=1}^{M} y_{o,c} \log\left(p_{o,c}\right)$$

„Loss Functions — ML Glossary documentation (ml-cheatsheet.readthedocs.io)“

$y_{o,c}$ nimmt 1 an, wenn die Beobachtung o der Klasse c ist, andernfalls 0. $P_{o,c}$ ist die vorhergesagte Wahrscheinlichkeit, dass die Beobachtung o aus der Klasse c stammt.

Rekurrente Neuronale Netze (RNN):

Standardmäßige künstliche neuronale Netze sind nicht in der Lage, die zeitliche Abhängigkeit zwischen Proben zu erfassen. Um solche historischen Datenpunkte einzubeziehen, wurden rekurrente neuronale Netze (RNN) angepasst, um zukünftige Werte durch Modellierung vergangener Informationen vorherzusagen. RNN ist eine Variante eines künstlichen neuronalen Netzwerks, das mit einem Mechanismus zum Umgang mit sequentiellen Daten oder Zeitreihen ausgestattet ist, in denen Datenpunkte eine zeitliche Korrelation aufweisen. Die RNN-Architektur verfügt über einen „Speicher“, der ihnen hilft, die Zustände früherer Eingaben zu speichern und sie zur Berechnung zukünftiger Prognosen der Sequenz zu verwenden.

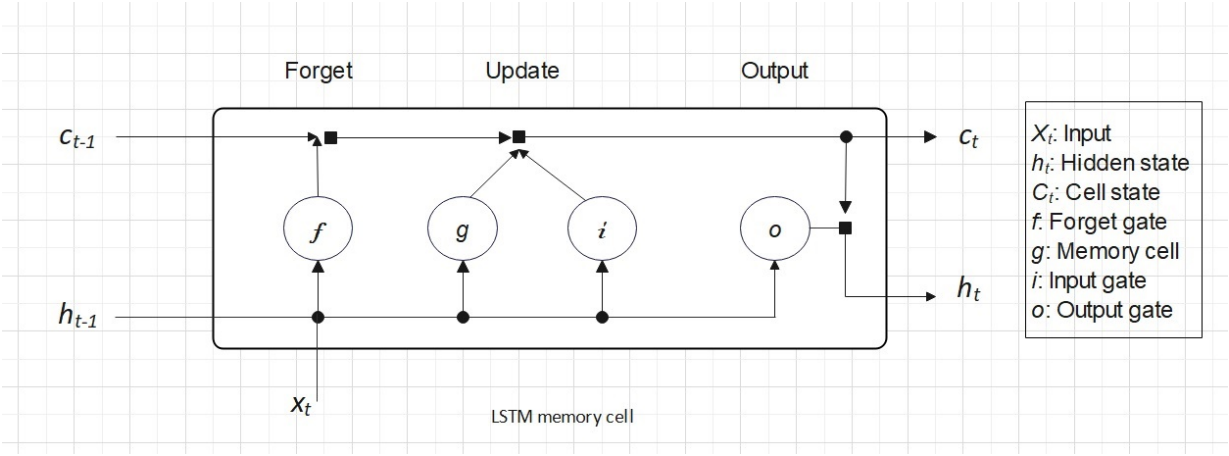


LSTM:

LSTM-Netzwerke sind eine Sonderform der RNN-Architektur, bei der eine Einheit (oder ein Neuron) zyklische Verbindungen enthält. Der Hauptunterschied zwischen einer LSTM-Entität und einer Standard-RNN-Entität besteht darin, dass LSTM aus sogenannten Gates besteht, die den Informationsfluss durch die Entität besser regulieren sollen. Das Hauptproblem von RNNs liegt in ihrer begrenzten Fähigkeit, längerfristige Abhängigkeiten zu lernen. Standard-RNNs können nicht lernen, wenn es aufgrund eines „verschwindenden“ oder „explodierenden“ Gradienten zu Zeitverzögerungen von mehr als 5 bis 10 diskreten Zeitschritten zwischen relevanten Eingabeereignissen und Zielsignalen kommt. Ein verschwindender Gradient tritt auf, wenn der Wert des Produkts der Ableitung abnimmt, bis sich irgendwann die partielle Ableitung der Verlustfunktion einem Wert nahe Null nähert, was ihre Fähigkeit, langfristige Beziehungen zu lernen, verringert. LSTM-Netzwerke gehen dieses Problem an, indem sie einen konstanten Fehlerfluss durch „Constant Error Carrousels“ (CECs) innerhalb spezieller Einheiten, sogenannter Zellen, erzwingen. Eine LSTM-Zelle enthält zwei Gatter, die lernen, auf den Fehlerfluss im CEC jeder Speicherzelle zuzugreifen. Das multiplikative Eingabegatter schützt den CEC vor Störungen durch irrelevante Eingaben. Ebenso schützt das multiplikative Ausgangsgatter andere Einheiten vor Störungen durch aktuell irrelevante Speicherinhalte. Diese zusätzlichen Tore ermöglichen es dem Netzwerk, langfristige Beziehungen zwischen

Datenpunkten effektiver zu lernen.

Aufgrund der geringeren Empfindlichkeit gegenüber der Zeitlücke eignen sich LSTM-Netzwerke besser für die sequentielle Datenanalyse als einfache RNNs. Zusätzlich zum verborgenen Zustand von RNNs verfügt die Architektur für einen LSTM typischerweise über eine Speicherzelle, ein Eingangs-Gate, ein Ausgangs-Gate und ein Oblivion-Gate, wie unten gezeigt:

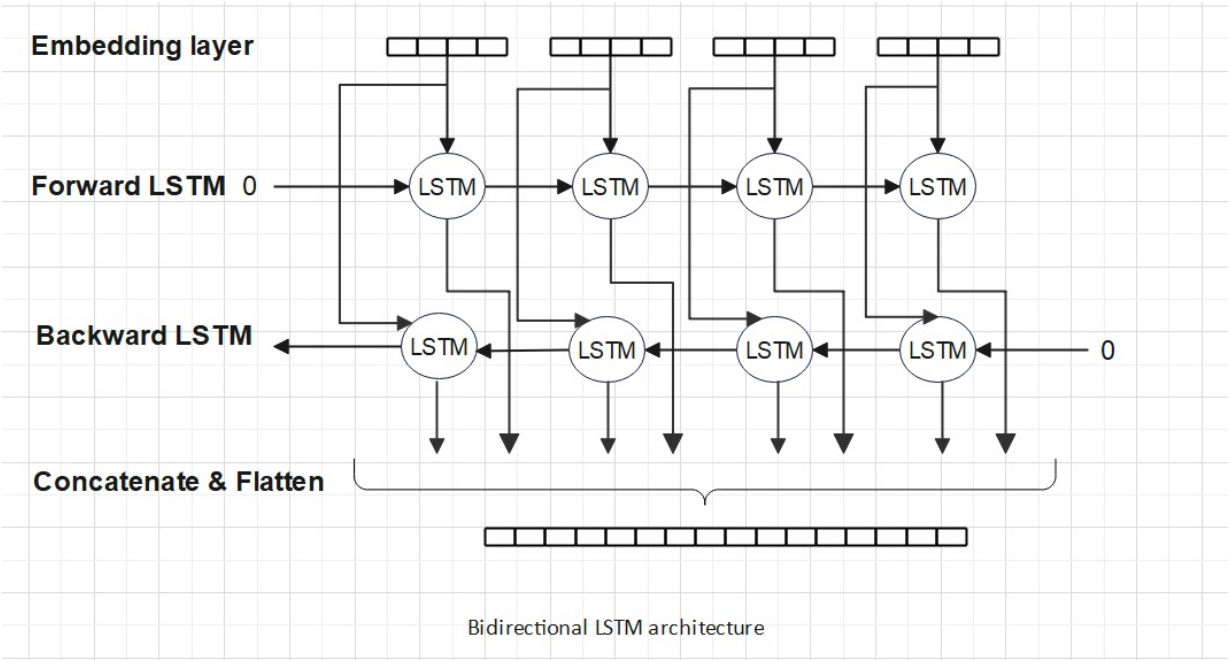


Dabei ist h der verborgene Zustand, der das Kurzzeitgedächtnis darstellt, c der Zellzustand, der das Langzeitgedächtnis darstellt, und x die Eingabe. Die Gewichtungen und Bias des Eingangsgatters steuern das Ausmaß, in dem ein neuer Wert in die Zelle fließt. In ähnlicher Weise steuern die Gewichtungen und Bias des Vergessens-Gates und des Ausgangs-Gates, welche Daten in der Zelle verbleiben und welche Daten zur Berechnung der Ausgangsaktivierung des LSTM-Blocks verwendet werden.

Im Gegensatz zum Standard-LSTM fließt die Eingabe bei bidirektionalem LSTM in beide Richtungen, da wir eine zusätzliche Ebene zum Trainieren der Umkehrung der Eingabesequenz haben. bidirektionales lstm ist ein leistungsstarkes Werkzeug zur Modellierung der sequentiellen Abhängigkeiten zwischen Wörtern und Phrasen in beiden Richtungen der Sequenz. Da zum Trainieren der Daten zwei Ebenen verwendet werden, muss die Ausgabe beider zusammengeführt werden. Der Zusammenführungsmechanismus kann eine der folgenden Funktionen haben:

1. Summe
2. Multiplikation
3. Mittelung
4. Verkettung (Standard)

Hier wird eine Einbettungsschicht verwendet, um eine Folge von Wortindizes in Einbettungsvektoren abzubilden, sodass das Netzwerk mehr über die Beziehung zwischen Wörtern erfahren kann.



Sequenzvorhersage Übung

Für das Training des bidirektionalen LSTM-Modells verwenden Twitter Datensatz hier .
Wir müssen den Text zeilenweise trennen, damit wir mithilfe des Tokenizers Sequenzen aus dem Text generieren können.
Lasst uns zunächst die Daten importieren und einen Blick darauf werfen:

In [8]:
```
import pandas as pd
import numpy as np
import heapq
import matplotlib.pyplot as plt
from nltk.tokenize import word_tokenize
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Activation
from keras.layers import LSTM


from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout
from keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.text import Tokenizer


#path ='text.txt'
path = 'en_US.twitter.txt'
# read and convert from one block text into a list of lines
with open(path, encoding='utf-8') as f:
     lines = [line.rstrip('\n') for line in f]


text =lines[:5000]
```

In [16]:
```
text[:5]
```

Out[16]:
```
['How are you? Btw thanks for the RT. You gonna be in DC anytime soon? Love to see you. Been way, way too long.',
 "When you meet someone special... you'll know. Your heart will beat more rapidly and you'll smile for no reason.",
 "they've decided its more fun if I don't.",
 'So Tired D; Played Lazer Tag & Ran A LOT D; Ughh Going To Sleep Like In 5 Minutes ;)',
 'Words from a complete stranger! Made my birthday even better :)']
```

Der erste Schritt besteht darin, die Wörter in den Texten zu tokenisieren, was bedeutet, den Text in eine Folge von ganzen Zahlen umzuwandeln (wobei jede ganze Zahl der Index eines Tokens in einem Wörterbuch ist). Dann müssen wir aus unserem Text ein n_gram-Modell erstellen. Ein n-Gramm ist eine zusammenhängende Folge von n Elementen aus einer bestimmten Text- oder Sprachprobe. Bei der Sprachmodellierung werden Unabhängigkeitsannahmen getroffen, sodass jedes Wort nur von den letzten n − 1 Wörtern abhängt. Dies hilft dem LSTM-Modell, den Kontext zu verstehen und die Beziehungen zwischen Wörtern zu lernen.

In [32]:
```
#oov_token: if given, it will be added to word_index and used to replace out-of-vocabulary words during text_to_sequence calls
# For those words which are in the vocab
#the first step is to tokenize the words in the texts ,
#which means turning the text  into  a sequence of integers
# (each integer being the index of a token in a dictionary).
# then we need to build n_gram model from our text. an n-gram  is a contiguous sequence of n items
# from a given sample of text or speech. in  language modeling,
# independence assumptions are made so that each word depends only on the last n - 1 words.
# this help lstm model to understand the context and to learn the relationships between words.
tokenizer = Tokenizer(oov_token='<oov>')
tokenizer.fit_on_texts(text['title'])
total_words = len(tokenizer.word_index) + 1
input_sequences = []
for line in text['title']:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

print("Total input sequences: ", len(input_sequences))
```

Die Eingabesequenzen müssen dieselbe Länge haben, daher verwenden wir die Auffülltechnik, um jeder Zeichenfolge Nullen hinzuzufügen, sodass sie dieselbe Länge wie die längste Sequenz in der Liste haben.

In [10]:
```
# sequence padding
from tensorflow.keras.preprocessing.sequence import pad_sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
input_sequences[1]
```

Out[10]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 54, 22,  6])
```

Als nächstes müssen wir Eingabedaten des Modells vorbereiten. Hier verwenden wir das letzte Wort in jeder Sequenz als das vom Modell vorherzusagende Wort, während der Rest für das Training verwendet wird.

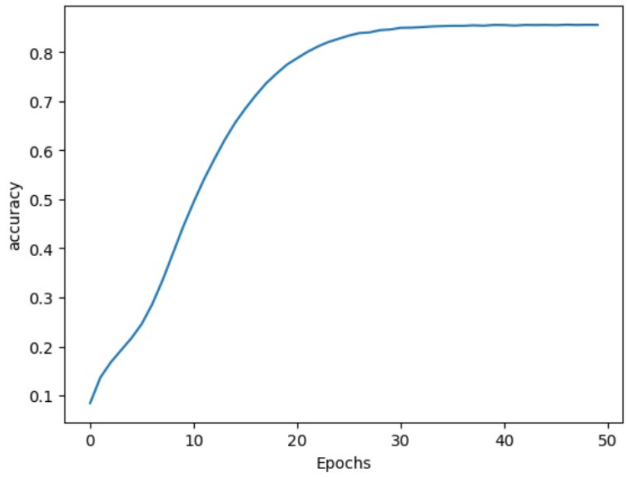
In [11]:
```
import tensorflow as tf
xs, labels = input_sequences[:,:-1],input_sequences[:,-1]
ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

Wir werden hier bidirektionales LSTM verwenden, das Sequenzen in beide Richtungen lernt. Eine Einbettungsebene wird hinzugefügt, um Eingabewörter in dichte Vektoren abzubilden. Diese Schicht ist sehr wichtig, da sie es dem Netzwerk ermöglicht, mehr über die Beziehung zwischen Eingaben zu erfahren und die Daten effizienter zu verarbeiten. Jedes Wort in unserem Modell hat einen Einbettungsvektor der Größe 100.

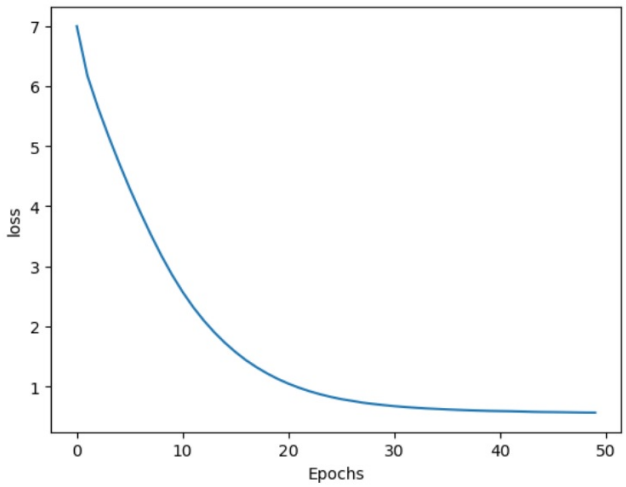
Um die Trainingsleistung des Modells zu überprüfen, zeichnen wir Genauigkeits- und Verlustwerte auf und prüfen, ob das Modell eine gute Lernleistung aufweist. Werfen Sie einen Blick auf Trainingsgenauigkeit und -verlust

In [30]:
```
# we will use here bidirectional LSTM, which learn sequences in both directions.
# An embedding layer is added to map input words into dense vectors. this layer is very important since
# it allows the network to learn more about the relationship between inputs and to process the data more efficiently.
# each word in our model will have an embedding vector of size 100.
from tensorflow.keras.optimizers import Adam
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=50, verbose=1)
#print model.summary()
print(model)
```

In [ ]:
```
# plotting model accuracy and loss
import matplotlib.pyplot as plt
def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.show()
plot_graphs(history, 'accuracy')
```



In [41]:
```
plot_graphs(history, 'loss')
```



Wir können das Modell speichern und jederzeit zur Vorhersage von Wörtern verwenden:

In [2]:
```
from keras.models import load_model
!mkdir -p saved_model
model.save('saved_model/my_model_twitter')
#del model  # deletes the existing model

# relaod the saved model
saved_model2 = load_model('saved_model/my_model_twitter')
#new_model.evaluate(test_images, test_labels, verbose=2)# to evaluate the model with new data
```

Um das nächste Wort eines Textes vorherzusagen, müssen wir es zunächst tokenisieren und in eine aufgefüllte Sequenz umwandeln. Jetzt werden wir unser Modell anhand neuer Daten testen und sehen, wie gut das Modell das nächste Wort basierend auf einem Starttext vorschlagen kann.

In [35]:
```
#Predicting next word of title
tokenizer = Tokenizer(oov_token='<oov>')
tokenizer.fit_on_texts(text)
seed_text = "it seem like"
seed_text_2= "we've decided"
num_words = 2

for _ in range(num_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = saved_model2.predict(token_list, verbose=0)
    class_x =np.argmax(predicted,axis=1)
    output_word = ""
    for word, index in tokenizer.word_index.items():
        if index == class_x:
            output_word = word
            break
    seed_text += " " + output_word
print(seed_text)
```

```
it seem like bat can
```


```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 8,
   "id": "de4a688e",
   "metadata": {},
   "outputs": [],
   "source": [
    "import pandas as pd\n",
    "import numpy as np\n",
    "import heapq\n",
    "import matplotlib.pyplot as plt\n",
    "from nltk.tokenize import word_tokenize\n",
    "from keras.models import Sequential, load_model\n",
    "from keras.layers.core import Dense, Activation\n",
    "from keras.layers import LSTM\n",
    "\n",
    "\n",
    "from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional, Dropout\n",
    "from keras.optimizers import RMSprop\n",
    "from tensorflow.keras.preprocessing.text import Tokenizer\n",
    "\n",
    "\n",
    "#path ='text.txt'\n",
    "path = 'en_US.twitter.txt'\n",
    "# read and convert from one block text into a list of lines \n",
    "with open(path, encoding='utf-8') as f:\n",
    "     lines = [line.rstrip('\\n') for line in f]\n",
    "#data = open(path, encoding='utf-8').read().lower()\n",
    "#text = pd.read_csv('medium_data.csv')\n",
    "#data# = data[:1000000]\n",
    "\n",
    "#text['title'] = text['title'].apply(lambda x: x.replace(u'\\xa0',u' '))\n",
    "#text['title'] = text['title'].apply(lambda x: x.replace('\\u200a',' '))\n",
    "\n",
    "text =lines[:5000]"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 16,
   "id": "9927b097",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "['How are you? Btw thanks for the RT. You gonna be in DC anytime soon? Love to see you. Been way, way too long.',\n",
       " \"When you meet someone special... you'll know. Your heart will beat more rapidly and you'll smile for no reason.\",\n",
       " \"they've decided its more fun if I don't.\",\n",
       " 'So Tired D; Played Lazer Tag & Ran A LOT D; Ughh Going To Sleep Like In 5 Minutes ;)',\n",
       " 'Words from a complete stranger! Made my birthday even better :)']"
      ]
     },
     "execution_count": 16,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "text[:5]\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "76b1ef0a",
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": 9,
   "id": "8845a829",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
```

```
     "output_type": "stream",
     "text": [
      "Total input sequences:  58427\n"
     ]
    }
   ],
   "source": [
    "#oov_token: if given, it will be added to word_index and used to replace out-of-vocabulary words during text_to_sequenc
    "tokenizer = Tokenizer(oov_token='<oov>') \n",
    "tokenizer.fit_on_texts(text)\n",
    "total_words = len(tokenizer.word_index) + 1\n",
    "input_sequences = []\n",
    "for line in text:\n",
    "    token_list = tokenizer.texts_to_sequences([line])[0]  \n",
    "    for i in range(1, len(token_list)):\n",
    "        n_gram_sequence = token_list[:i+1]\n",
    "        input_sequences.append(n_gram_sequence)\n",
    "\n",
    "print(\"Total input sequences: \", len(input_sequences))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 10,
   "id": "69aa04a9",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,\n",
       "        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, 54, 22,  6])"
      ]
     },
     "execution_count": 10,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "# sequence padding\n",
    "from tensorflow.keras.preprocessing.sequence import pad_sequences\n",
    "max_sequence_len = max([len(x) for x in input_sequences])\n",
    "input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))\n",
    "input_sequences[1]"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 11,
   "id": "b83631a9",
   "metadata": {},
   "outputs": [],
   "source": [
    "import tensorflow as tf\n",
    "xs, labels = input_sequences[:,:-1],input_sequences[:,-1]\n",
    "ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 30,
   "id": "00ce7a38",
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optim
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Epoch 1/50\n",
      "1826/1826 [==============================] - 192s 104ms/step - loss: 7.3348 - accuracy: 0.0382\n",
      "Epoch 2/50\n",
      "1826/1826 [==============================] - 188s 103ms/step - loss: 6.6876 - accuracy: 0.0637\n",
      "Epoch 3/50\n",
      "1826/1826 [==============================] - 194s 106ms/step - loss: 6.2560 - accuracy: 0.0847\n",
      "Epoch 4/50\n",
      "1826/1826 [==============================] - 201s 110ms/step - loss: 5.8215 - accuracy: 0.1013\n",
      "Epoch 5/50\n",
      "1826/1826 [==============================] - 196s 107ms/step - loss: 5.3644 - accuracy: 0.1205\n",
      "Epoch 6/50\n",
      "1826/1826 [==============================] - 196s 107ms/step - loss: 4.9059 - accuracy: 0.1501\n",
      "Epoch 7/50\n",
      "1826/1826 [==============================] - 196s 107ms/step - loss: 4.4605 - accuracy: 0.1949\n",
      "Epoch 8/50\n",
      "1826/1826 [==============================] - 191s 104ms/step - loss: 4.0379 - accuracy: 0.2496\n",
      "Epoch 9/50\n",
      "1826/1826 [==============================] - 195s 107ms/step - loss: 3.6459 - accuracy: 0.3065\n",
      "Epoch 10/50\n",
      "1826/1826 [==============================] - 195s 107ms/step - loss: 3.2921 - accuracy: 0.3619\n",
      "Epoch 11/50\n",
      "1826/1826 [==============================] - 195s 107ms/step - loss: 2.9712 - accuracy: 0.4153\n",
      "Epoch 12/50\n",
      "1826/1826 [==============================] - 189s 104ms/step - loss: 2.6866 - accuracy: 0.4635\n",
      "Epoch 13/50\n",
      "1826/1826 [==============================] - 191s 104ms/step - loss: 2.4310 - accuracy: 0.5111\n",
      "Epoch 14/50\n",
      "1826/1826 [==============================] - 192s 105ms/step - loss: 2.2014 - accuracy: 0.5534\n",
```

```
      "Epoch 15/50\n",
      "1826/1826 [==============================] - 193s 106ms/step - loss: 1.9942 - accuracy: 0.5915\n",
      "Epoch 16/50\n",
      "1826/1826 [==============================] - 194s 106ms/step - loss: 1.8091 - accuracy: 0.6284\n",
      "Epoch 17/50\n",
      "1826/1826 [==============================] - 184s 101ms/step - loss: 1.6453 - accuracy: 0.6604\n",
      "Epoch 18/50\n",
      "1826/1826 [==============================] - 187s 103ms/step - loss: 1.4963 - accuracy: 0.6901\n",
      "Epoch 19/50\n",
      "1826/1826 [==============================] - 189s 104ms/step - loss: 1.3686 - accuracy: 0.7162\n",
      "Epoch 20/50\n",
      "1826/1826 [==============================] - 183s 100ms/step - loss: 1.2519 - accuracy: 0.7398\n",
      "Epoch 21/50\n",
      "1826/1826 [==============================] - 187s 103ms/step - loss: 1.1449 - accuracy: 0.7643\n",
      "Epoch 22/50\n",
      "1826/1826 [==============================] - 191s 105ms/step - loss: 1.0536 - accuracy: 0.7802\n",
      "Epoch 23/50\n",
      "1826/1826 [==============================] - 191s 104ms/step - loss: 0.9707 - accuracy: 0.7987\n",
      "Epoch 24/50\n",
      "1826/1826 [==============================] - 193s 106ms/step - loss: 0.8942 - accuracy: 0.8152\n",
      "Epoch 25/50\n",
      "1826/1826 [==============================] - 182s 100ms/step - loss: 0.8305 - accuracy: 0.8280\n",
      "Epoch 26/50\n",
      "1826/1826 [==============================] - 186s 102ms/step - loss: 0.7719 - accuracy: 0.8402\n",
      "Epoch 27/50\n",
      "1826/1826 [==============================] - 189s 103ms/step - loss: 0.7218 - accuracy: 0.8493\n",
      "Epoch 28/50\n",
      "1826/1826 [==============================] - 182s 100ms/step - loss: 0.6743 - accuracy: 0.8595\n",
      "Epoch 29/50\n",
      "1826/1826 [==============================] - 184s 101ms/step - loss: 0.6351 - accuracy: 0.8675\n",
      "Epoch 30/50\n",
      "1826/1826 [==============================] - 184s 101ms/step - loss: 0.6017 - accuracy: 0.8729\n",
      "Epoch 31/50\n",
      "1826/1826 [==============================] - 187s 103ms/step - loss: 0.5667 - accuracy: 0.8800\n",
      "Epoch 32/50\n",
      "1826/1826 [==============================] - 193s 106ms/step - loss: 0.5396 - accuracy: 0.8844\n",
      "Epoch 33/50\n",
      "1826/1826 [==============================] - 193s 106ms/step - loss: 0.5153 - accuracy: 0.8894\n",
      "Epoch 34/50\n",
      "1826/1826 [==============================] - 191s 105ms/step - loss: 0.4966 - accuracy: 0.8916\n",
      "Epoch 35/50\n",
      "1826/1826 [==============================] - 192s 105ms/step - loss: 0.4777 - accuracy: 0.8943\n",
      "Epoch 36/50\n",
      "1826/1826 [==============================] - 185s 101ms/step - loss: 0.4586 - accuracy: 0.8982\n",
      "Epoch 37/50\n",
      "1826/1826 [==============================] - 186s 102ms/step - loss: 0.4454 - accuracy: 0.9010\n",
      "Epoch 38/50\n",
      "1826/1826 [==============================] - 189s 104ms/step - loss: 0.4365 - accuracy: 0.9006\n",
      "Epoch 39/50\n",
      "1826/1826 [==============================] - 188s 103ms/step - loss: 0.4242 - accuracy: 0.9036\n",
      "Epoch 40/50\n",
      "1826/1826 [==============================] - 188s 103ms/step - loss: 0.4123 - accuracy: 0.9050\n",
      "Epoch 41/50\n",
      "1826/1826 [==============================] - 191s 105ms/step - loss: 0.4068 - accuracy: 0.9049\n",
      "Epoch 42/50\n",
      "1826/1826 [==============================] - 189s 104ms/step - loss: 0.3989 - accuracy: 0.9067\n",
      "Epoch 43/50\n",
      "1826/1826 [==============================] - 194s 107ms/step - loss: 0.3861 - accuracy: 0.9085\n",
      "Epoch 44/50\n",
      "1826/1826 [==============================] - 193s 106ms/step - loss: 0.3835 - accuracy: 0.9083\n",
      "Epoch 45/50\n",
      "1826/1826 [==============================] - 189s 104ms/step - loss: 0.3818 - accuracy: 0.9087\n",
      "Epoch 46/50\n",
      "1826/1826 [==============================] - 189s 104ms/step - loss: 0.3734 - accuracy: 0.9100\n",
      "Epoch 47/50\n",
      "1826/1826 [==============================] - 188s 103ms/step - loss: 0.3659 - accuracy: 0.9106\n",
      "Epoch 48/50\n",
      "1826/1826 [==============================] - 191s 104ms/step - loss: 0.3625 - accuracy: 0.9109\n",
      "Epoch 49/50\n",
      "1826/1826 [==============================] - 194s 106ms/step - loss: 0.3641 - accuracy: 0.9095\n",
      "Epoch 50/50\n",
      "1826/1826 [==============================] - 192s 105ms/step - loss: 0.3610 - accuracy: 0.9104\n",
      "<keras.engine.sequential.Sequential object at 0x0000015111BEADF0>\n"
     ]
    }
   ],
   "source": [
    "# we will use here bidirectional LSTM, which learn sequences in both directions.\n",
    "# An embedding layer is added to map input words into dense vectors. this layer is very important since\n",
    "# it allows the network to learn more about the relationship between inputs and to process the data more efficiently.\n
    "# each word in our model will have an embedding vector of size 100. \n",
    "from tensorflow.keras.optimizers import Adam\n",
    "model = Sequential()\n",
    "model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))\n",
    "model.add(Bidirectional(LSTM(150)))\n",
    "model.add(Dense(total_words, activation='softmax'))\n",
    "adam = Adam(lr=0.01)\n",
    "model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])\n",
    "history = model.fit(xs, ys, epochs=50, verbose=1)\n",
    "#print model.summary()\n",
    "print(model)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "8cc351b1",
   "metadata": {},
   "outputs": [],
   "source": [
    "# plotting model accuracy and loss\n",
    "import matplotlib.pyplot as plt\n",
    "def plot_graphs(history, string):\n",
    "    plt.plot(history.history[string])\n",
```

```
    "    plt.xlabel(\"Epochs\")\n",
    "    plt.ylabel(string)\n",
    "    plt.show()\n",
    "plot_graphs(history, 'accuracy')   "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "4ad125a9",
   "metadata": {},
   "outputs": [],
   "source": [
    "plot_graphs(history, 'loss')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "aaa39ea2",
   "metadata": {},
   "outputs": [],
   "source": [
    "from keras.models import load_model\n",
    "!mkdir -p saved_model    \n",
    "model.save('saved_model/my_model_twitter')  \n",
    "#del model  # deletes the existing model\n",
    "    \n",
    "# relaod the saved model \n",
    "saved_model2 = load_model('saved_model/my_model_twitter')\n",
    "#new_model.evaluate(test_images, test_labels, verbose=2)# to evaluate the model with new data "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 35,
   "id": "79ab4417",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "it seem like bat can\n"
     ]
    }
   ],
   "source": [
    "#Predicting next word of title\n",
    "tokenizer = Tokenizer(oov_token='<oov>') \n",
    "tokenizer.fit_on_texts(text)\n",
    "seed_text = \"it seem like\"\n",
    "seed_text_2= \"we've decided\"\n",
    "num_words = 2\n",
    "  \n",
    "for _ in range(num_words):\n",
    "    token_list = tokenizer.texts_to_sequences([seed_text])[0]\n",
    "    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')\n",
    "    predicted = saved_model2.predict(token_list, verbose=0)\n",
    "    class_x =np.argmax(predicted,axis=1)\n",
    "    output_word = \"\"\n",
    "    for word, index in tokenizer.word_index.items():\n",
    "        if index == class_x:\n",
    "            output_word = word\n",
    "            break\n",
    "    seed_text += \" \" + output_word\n",
    "print(seed_text)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 15,
   "id": "a90ce91b",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "[[None, None]]"
      ]
     },
     "execution_count": 15,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "token_list"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 24,
   "id": "546abfc2",
   "metadata": {},
   "outputs": [
    {
     "ename": "KeyError",
     "evalue": "'accuracy'",
     "output_type": "error",
     "traceback": [
      "\u001b[1;31m---------------------------------------------------------------------------\u001b[0m",
      "\u001b[1;31mKeyError\u001b[0m                                  Traceback (most recent call last)",
      "Cell \u001b[1;32mIn[24], line 8\u001b[0m\n\u001b[0;32m      6\u001b[0m     plt\u001b[38;5;241m.\u001b[39mylabel(strin
```

```
       "Cell \u001b[1;32mIn[24], line 4\u001b[0m, in \u001b[0;36mplot_graphs\u001b[1;34m(history, string)\u001b[0m\n\u001b[0;
       "\u001b[1;31mKeyError\u001b[0m: 'accuracy'"
      ]
     }
    ],
    "source": [
     " "
    ]
   },
   {
    "cell_type": "code",
    "execution_count": null,
    "id": "7f8cb838",
    "metadata": {},
    "outputs": [],
    "source": []
   }
  ],
  "metadata": {
   "kernelspec": {
    "display_name": "Python 3 (ipykernel)",
    "language": "python",
    "name": "python3"
   },
   "language_info": {
    "codemirror_mode": {
     "name": "ipython",
     "version": 3
    },
    "file_extension": ".py",
    "mimetype": "text/x-python",
    "name": "python",
    "nbconvert_exporter": "python",
    "pygments_lexer": "ipython3",
    "version": "3.9.13"
   }
  },
  "nbformat": 4,
  "nbformat_minor": 5
 }
```


Chatbots

Die Weiterentwicklung der KI-Technologie hat vielen Branchen, einschließlich der Wirtschaft, viele Vorteile gebracht, indem sie von traditionellen zu digitalen Plattformen für Transaktionen und Kommunikation mit Kunden übergegangen ist. Chatbots sind eine vielseitige KI-Technologie. Einige Beispiele für Chatbot-Technologie sind virtuelle Assistenten wie Amazons Alexa und Google Assistant sowie Messaging-Apps wie WeChat und Facebooks Messenger.

Eine Definition von Chatbot ist, dass es sich um eine Softwareanwendung oder ein Computerprogramm handelt, das in der Lage ist, ein Gespräch mit einem Benutzer in natürlicher Sprache zu führen, die Bedeutung und den Kontext seiner Eingaben zu verstehen und entsprechend zu antworten. Das Hauptziel dieser interessanten Technologie besteht darin, das Verhalten eines Menschen als Gesprächspartner zu simulieren. Chatbot-Systeme befinden sich noch in der Entwicklung und müssen daher kontinuierlich optimiert und getestet werden.

Chatbot kann mithilfe von drei Technologien implementiert werden:

1. Künstliches neuronales Netzwerk hier
2. Wörterbeutel
3. Lemmatisierung

Bevor wir uns mit der Erstellung von Chatbots in Python befassen, müssen wir diese Begriffe im Detail verstehen.

Wörterbeutel:

Das Bag-of-Words-Modell ist eine beliebte Darstellungsmethode in natürlicher Sprache, die einen Korpus durch eine Reihe seiner Wörter mit ihren Häufigkeiten im Text symbolisiert. Dieses Modell berücksichtigt nicht die Grammatik oder die Position der Wörter, sondern nur ihr Vorkommen. Das Bag-of-Words-Modell wird zur Klassifizierung von Dokumenten verwendet, wobei die Worthäufigkeit ein Trainingsmerkmal des Klassifikators ist.

Lemmatisierung:

Lemmatisierung ist ein algorithmischer Prozess, um Wörter im Text in ihre Lemma- oder Wörterbuchform zurückzuführen. Mit anderen Worten: Es identifiziert die Bedeutung des Wortes in der Sprache und den Gesamtkontext, während es sich nicht darum kümmert, den grammatikalischen Zustand der Wörter zu erkennen. Beispielsweise wird das Wort „went" durch Lemmatisierung in das Wort „go" umgewandelt.

Anwendungsfälle

Chatbots werden in Dialogsystemen zu unterschiedlichen Zwecken eingesetzt, unter anderem zur Kundenbetreuung, zur Weiterleitung von Anfragen oder zur Informationsbeschaffung. Einige Chatbot-Anwendungen verwenden umfangreiche Wortklassifikatoren und Prozessoren für natürliche Sprache, während andere Anwendungen einfach nach allgemeinen Schlüsselwörtern im Text suchen und Antworten mithilfe von Phrasen generieren, die aus einer zugehörigen Bibliothek stammen.

Chatbots Übung

Wir werden hier Intents-JSON-Daten verwenden hier, um einen einfachen Chatbot zu erstellen. In diesem Chatbot trainieren wir das neuronale Netzwerk Artificail, um die Beziehung zwischen Wortsequenzen und ihren Tags zu lernen, da der Datensatz Tags für jeden Dateneintrag enthält. Bei diesem Ansatz generieren wir die Antwort nicht, indem wir den Kontext des Eingabetexts verstehen, sondern indem wir das ANN-Mo-

dell dazu bringen, das Tag des Eingabetexts vorherzusagen. Nachdem das Tag der Eingabedaten vorhergesagt wurde, kann aus den jedem Tag beigefügten Beispielantworten eine Antwort ausgewählt werden.

In [13]:
```
import json
import random
import string
import nltk
import numpy as np

#reading json file
file= open('intents.json').read()

chats = json.loads(file)

len(chats['intents'])
chats
```

```
    'responses': ['Happy to help!', 'Any time!', 'My pleasure'],
    'context': ['']},
   {'tag': 'noanswer',
    'patterns': [],
    'responses': ["Sorry, can't understand you",
     'Please give me more info',
     'Not sure I understand'],
    'context': ['']},
   {'tag': 'options',
    'patterns': ['How you could help me?',
     'What help you provide?',
     'How you can be helpful?'],
    'responses': ['I can guide you through Adverse management problems, order tracking, person to be contacted and Department
related queries',
     'I can provide support related to following problems technical query,management related query,order related query,trackin
g related query,procurement query,outsourcing problem,manufacturing delay,'],
    'context': ['']},
```

Der erste Schritt hier besteht darin, dass wir die Chat-Skripte mit nltk.word_tokenize kodieren. Einige Daten müssen bereinigt werden, um Tags und Satzzeichen zu entfernen, da sie dem Formular keine nützlichen Informationen liefern können.

In [2]:
```
# identifying features and target.
# tokenizing words is achieved  using nltk.word_tokenize.
# data_X represents features and data_Y represents tags of the patterns.
#  words list contains all tokens , classes contains the corresponding tags, both are sorted und duplicate-cleaned.
#The first step here is to prepare our input data by identifying features represented by chats
# and goals represented by tags. Additionally, we need to tokenize the chat texts with nltk.word_tokenize.
#Some data cleansing is required to remove the tags and punctuation as these cannot provide meaningful information
#to the model.
nltk.download('punkt')
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()
ignore = ['?']
words =[]
classes = []
docs = []
data_Y =[]

for chat in chats['intents']:
    for patt in chat['patterns']:
        tokens = nltk.word_tokenize(patt)#tokenize the pattern
        words.extend(tokens)##add tokens to words list
        docs.append((tokens, chat['tag'])) ## append the pattern to
        data_Y.append(chat['tag'])

    #add tags to classes list
        if chat['tag'] not in classes:
            classes.append(chat['tag'])
```

Das Entstammen von Wörtern ist ein wichtiger Schritt bei der Ableitung von Wortstämmen und deren Umwandlung in Kleinbuchstaben, da sie dem Modell dabei helfen können, das Wort auch dann zu erkennen, wenn es in unterschiedlichen Konjugationen vorkommt.

In [17]:
```
#stemming all wordsand convert them to lower cases, remove punctuations
import nltk
nltk.download('punkt')
from nltk.stem.lancaster import LancasterStemmer
#from nltk.stem import WordNetLemmatizer
stemmer =  LancasterStemmer()
words = [stemmer.stem(w.lower()) for w in words if w not in ignore]
words = sorted(list(set(words)))

# remove duplicate classes
classes = sorted(list(set(classes)))

print (len(docs), "documents")
print (len(classes), "classes", classes)
print (len(words), "unique stemmed words", words)
```

```
121 documents
43 classes ['HR_related_problem', 'Location', 'Weather', 'about', 'appointment status', 'cabin', 'check_leave', 'commission',
'competitors_in_market', 'configuration', 'connect_people', 'cost_lowering', 'customer_satisfaction', 'domain', 'email_id', 'fa
ctors_impacting_sale', 'forgot_password', 'gadgets', 'goodbye', 'greeting', 'highest_grossing', 'hours', 'invalid', 'key_custom
ers', 'leave', 'maintainence', 'manufacturing_problems', 'missing_id', 'name', 'noans', 'options', 'order_components', 'order_t
racking', 'predict_delay', 'predict_performance', 'project_handling_queries', 'search_department', 'search_person_by_id', 'solv
```

Um das ANN-Modell zu trainieren, müssen wir unsere Eingabedaten einrichten, indem wir Merkmale und Ziele definieren, die durch Muster bzw. Tags dargestellt werden. Um Eingabetext in Ganzzahlen umzuwandeln, erstellen wir aus dem Datensatz ein Bag-of-Words-Modell. Ein Vokabular ist eine Textdarstellung, die das Vorkommen von Wörtern in einem Dokument beschreibt.

```
training = []
#output = []
# create an empty array for output
output_empty = [0] * len(classes)

# create training set, bag of words for each sentence
for doc in docs:
    # initialize bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # stemming each word
    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]
    # create bag of words array
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is '1' for current tag and '0' for rest of other tags
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
print(len(classes))
# shuffling features and turning it into np.array
random.shuffle(training)
training = np.array(training)

# creating training lists
train_x = list(training[:,0])
train_y = list(training[:,1])
```

```
43
```

Wir werden ein sequentielles Modell von Keras erstellen. Die Eingabe in dieses Netzwerk ist das Array train_X . Das Modell besteht aus drei Schichten, wobei die erste Schicht 128 Neuronen, die zweite Schicht 64 Neuronen und die letzte Schicht die gleiche Anzahl Neuronen wie Klassen aufweist. Um die richtigen Gewichte zu erreichen, wird der Adam-Optimierer verwendet. Die Verlustfunktion ist eine kategoriale Kreuzentropiefunktion, da wir ein KNN-Klassifizierungsmodell trainieren.

In [5]:
```
#building neural network model,we'll create Keras' Sequential model.
#The input to this network will be the array train_X
# the model has 3 layers with the first having 128 neurons, the second having 64 neurons, and the last layer having the same numb
# Next, to reach the correct weights, Adam optimizer is used. loss function is  categorical cross-entropy function.
# And, the metric  to evaluate the model is 'accuracy'.
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
model = Sequential()
# the number of units(neurons) could affect the resulted training accuracy
model.add(Dense(128, input_shape =(len(train_x[0]),), activation ='relu'))#input layer
model.add(Dropout(0.5))#dropout layer
model.add(Dense(64, activation ='relu'))#hidden layer
model.add(Dropout(0.5))
# output layer, classification model(activation function: softmax)
model.add(Dense(len(train_y[0]), activation ='softmax'))
adam =tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss= 'categorical_crossentropy',
              optimizer =adam,
              metrics=['accuracy'])
print(model.summary)
model.fit(x=train_x, y=train_y, epochs=1000, batch_size=128)
```

```
Epoch 985/1000
1/1 [==============================] - 0s 0s/step - loss: 0.1326 - accuracy: 0.9917
Epoch 986/1000
1/1 [==============================] - 0s 3ms/step - loss: 0.0611 - accuracy: 0.9669
Epoch 987/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.0817 - accuracy: 0.9587
Epoch 988/1000
```

Es ist Zeit, unser Modell zu bewerten. Wir beginnen mit der Vorbereitung Ihrer Eingabedaten. Dazu gehört die Wort-Tokenisierung und die Erstellung des Bag_of_words-Modells:

In [17]:
```
# model evaluation with unknown data
# chat preparation functions
def sent_clean(sent):
    # tokenizing the pattern
    tokens = nltk.word_tokenize(sent)
    # stemming each word
    sent_words = [stemmer.stem(word.lower()) for word in tokens]
    return sent_words

# returning bag of words of the pattern
def bw(sent, words):
    # tokenizing the pattern
    sent_words = sent_clean(sent)
    # create bag of words
    bag = [0]*len(words)
    for s in sent_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1


    return(np.array(bag))
```

In [18]:
```
# classify the pattern with a tag, and generate the appropriate response
error_threshold = 0.30
def classify(sent):
    # predict class probabilities from the model
    #tag_prob = model.predict(np.array([bw(sent, words)]))[0]# for sequential model
    tag_prob = model.predict([bw(sent, words)])[0] # for dnn model
    # filter out predictions below a threshold
    results = [[i,r] for i,r in enumerate(tag_prob) if r > error_threshold]
    # sort probabilities in descending order
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append((classes[r[0]], r[1]))# get the tag of index r[0] und prob_value is r[1]
    # return tuple of intent and probability
    return return_list

def response(sentence):
    results = classify(sentence)
    # if we have a classification then find the matching intent tag
    if results:
        # loop as long as there are matches to process
        while results:
            for chat in chats['intents']:
                # find a tag matching the first result (highst probability)
                if chat['tag'] == results[0][0]:
                    # print out a random response of the corresponding tag
                    return print(random.choice(chat['responses']))

            results.pop(0)
```

In [19]:
```
print('enter 0 if you dont want to chat with this Chatbot')
while True:

    message = input('')
    if message==0:
        break
    res = response(message)
    print(res)
```

```
enter 0 if you dont want to chat with this Chatbot
hi
Good to see you again
None
where is my order?
Please enter with order ID
None
123
please elaborate your question
None
I want to know the various departments in this company
The department are:Projects,IT,Production,OutSource
None
```


```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "4550aae0",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "{'intents': [{'tag': 'greeting',\n",
       "   'patterns': ['Hi there',\n",
       "    'How are you',\n",
       "    'Is anyone there?',\n",
       "    'Hey',\n",
       "    'Hola',\n",
       "    'Hello',\n",
       "    'Good day'],\n",
       "   'responses': ['Hello, thanks for asking',\n",
       "    'Good to see you again',\n",
       "    'Hi there, how can I help?'],\n",
       "   'context': ['']},\n",
       "  {'tag': 'goodbye',\n",
       "   'patterns': ['Bye',\n",
       "    'See you later',\n",
       "    'Goodbye',\n",
       "    'Nice chatting to you, bye',\n",
       "    'Till next time'],\n",
       "   'responses': ['See you!', 'Have a nice day', 'Bye! Come back again soon.'],\n",
       "   'context': ['']},\n",
       "  {'tag': 'thanks',\n",
       "   'patterns': ['Thanks',\n",
       "    'Thank you',\n",
       "    \"That's helpful\",\n",
       "    'Awesome, thanks',\n",
       "    'Thanks for helping me'],\n",
       "   'responses': ['Happy to help!', 'Any time!', 'My pleasure'],\n",
       "   'context': ['']},\n",
       "  {'tag': 'noanswer',\n",
       "   'patterns': [],\n",
       "   'responses': [\"Sorry, can't understand you\",\n",
       "    'Please give me more info',\n",
       "    'Not sure I understand'],\n",
       "   'context': ['']},\n",
       "  {'tag': 'options',\n",
       "   'patterns': ['How you could help me?',\n",
       "    'What help you provide?',\n",
       "```
```

```
        "    'How you can be helpful?'],\n",
        "   'responses': ['I can guide you through Adverse management problems, order tracking, person to be contacted and D
        "    'I can provide support related to following problems technical query,management related query,order related que
        "   'context': ['']},\n",
        "  {'tag': 'order_tracking',\n",
        "   'patterns': ['where is order with id 431B67?',\n",
        "    'track order 562B78',\n",
        "    'Where is my order with id 561A24?'],\n",
        "   'responses': ['Delayed', 'Dispatched', 'On the Way!'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'order_components',\n",
        "   'patterns': ['order id 345A23 comprises of?', 'List of components'],\n",
        "   'responses': ['order comprises of hardisk and bluetooth'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'missing_id',\n",
        "   'patterns': ['where is the order',\n",
        "    'where is my order',\n",
        "    'locate the order',\n",
        "    'Delivery date of order'],\n",
        "   'responses': ['Please enter with order ID'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'Location',\n",
        "   'patterns': ['find order location 32712',\n",
        "    'What is the Location of order 23A31?'],\n",
        "   'responses': ['It is in germany',\n",
        "    'It is in Berlin',\n",
        "    'It has reached china',\n",
        "    'It has reached Banglore'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'search_person_by_id',\n",
        "   'patterns': ['I want an appoitment with Manoj kumar',\n",
        "    'set an appointment with Sujata Nandi'],\n",
        "   'responses': ['Fixing an appointment.'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'appointment status',\n",
        "   'patterns': ['Is my appointment fixed?', 'Do I have an appointment today?'],\n",
        "   'responses': ['Yes', 'NO! Not Yet'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'check_leave',\n",
        "   'patterns': ['Is Sujata Nandi on leave?', 'Is Manish Kumar on leave?'],\n",
        "   'responses': ['No Not On Leave', 'Yes On Leave'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'cost_lowering',\n",
        "   'patterns': ['Cost Lowering changes to be made?',\n",
        "    'what changes could lower cost?'],\n",
        "   'responses': ['Increase Transportation capacity utilisation,Increase supply chain velocity,Reduce order variabil
        "   'context': ['']},\n",
        "  {'tag': 'forgot_password',\n",
        "   'patterns': ['I forgot my Login password?',\n",
        "    'what to do when someone forgets Login password?',\n",
        "    'I forgot my Laptop password',\n",
        "    'Forgot Wifi password'],\n",
        "   'responses': ['Please enter your email id ,we will send a link on your email'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'email_id',\n",
        "   'patterns': ['abx@gmail.com', 'abc@kiit.ac.in'],\n",
        "   'responses': ['Link has been shared.Please change your password'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'manufacturing_problems',\n",
        "   'patterns': ['Find me a manufacturer nearby'],\n",
        "   'responses': ['The nearest manufacturer is Vietnam'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'search_department',\n",
        "   'patterns': ['I want to know the various departments in this company'],\n",
        "   'responses': ['The department are:Projects,IT,Production,OutSource'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'competitors_in_market',\n",
        "   'patterns': ['what are challenging events',\n",
        "    'what are the threats in the market'],\n",
        "   'responses': ['Recent news of Demonetisation & recession'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'key_customers',\n",
        "   'patterns': ['Our Target customers', 'who are your key customers'],\n",
        "   'responses': ['Our target customers are in the range of age 20-40'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'supplier_info',\n",
        "   'patterns': ['What information is shared with supplier?'],\n",
        "   'responses': ['Production Schedule,Delivery Schedule,Proxy information about cost'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'highest_grossing',\n",
        "   'patterns': ['What is the highest grossing product?'],\n",
        "   'responses': ['Laptop with touch sensors and 360 rotation'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'connect_people',\n",
        "   'patterns': ['I want to meet the head of HR/IT/Projects department',\n",
        "    'I want to meet head IT Rakesh sharma.'],\n",
        "   'responses': ['I will just check if he is available or on leave.'],\n",
        "   'context': ['search_attendance_database_name_post']},\n",
        "  {'tag': 'project_handling_queries',\n",
        "   'patterns': ['complaint has been raised for Insufficient Team Skills',\n",
        "    'Miscommunication Conflicts needs to be resolved',\n",
        "    'Risk Management issue has occured',\n",
        "    'skilled employees needed urgently'],\n",
        "   'responses': ['Please contact the project depatment'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'solve_problems',\n",
        "   'patterns': ['Lack of product clarity',\n",
        "    'the specifications of product is not clear to customer. what should we do'],\n",
        "   'responses': ['Please enter product id and customer id.specifications shall be sent to user'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'version_update',\n",
        "   'patterns': ['user needs the updated software version',\n",
        "    'user demands software updation'],\n",
        "   'responses': ['version updation request raised'],\n",
```

```
"   'context': ['']},\n",
"  {'tag': 'HR_related_problem',\n",
"   'patterns': ['problem related to Job design and analysis',\n",
"    'query based on Workforce planning',\n",
"    'Training and development issues are to be handled',\n",
"    'Compensation and benefits of the working employee',\n",
"    'Legal issues of department like accidents in the company'],\n",
"   'responses': ['Please Contact the HR team for this problem.'],\n",
"   'context': ['']},\n",
"  {'tag': 'factors_impacting_sale',\n",
"   'patterns': ['Impact on sale?', 'factors impacting sale this year?'],\n",
"   'responses': ['elections will impact our sale this year',\n",
"    'this year our sale might increase during durga puja',\n",
"    'this year sale might be impacted due to demonitisation'],\n",
"   'context': ['']},\n",
"  {'tag': 'predict_performance',\n",
"   'patterns': ['how have we improved our sale from last year?',\n",
"    'what is the profit this year?'],\n",
"   'responses': ['we have improved our sale by increasing our customers upto 2 lakh',\n",
"    'profit earned is 15%'],\n",
"   'context': ['']},\n",
"  {'tag': 'customer_satisfaction',\n",
"   'patterns': ['how was the customer response',\n",
"    'Is the customer happy?',\n",
"    'what was the customer feedback?'],\n",
"   'responses': ['Customer was happy and has given good rating'],\n",
"   'context': ['']},\n",
"  {'tag': 'maintainence',\n",
"   'patterns': ['maintainence related queries recorded'],\n",
"   'responses': ['lift is not working, projector misfuctioning'],\n",
"   'context': ['']},\n",
"  {'tag': 'gadgets',\n",
"   'patterns': ['what are the gadgets in stock?', 'which products we have?'],\n",
"   'responses': ['hardisk,pendrives, wireless bluetooth,Laptops,Gaming Accessories'],\n",
"   'context': ['']},\n",
"  {'tag': 'commission',\n",
"   'patterns': ['what is the Commission rate?',\n",
"    'Commission rate on each product?'],\n",
"   'responses': ['Commission rate is 5% on laptops,25%on bluetooth device,10% on Desktop'],\n",
"   'context': ['']},\n",
"  {'tag': 'invalid',\n",
"   'patterns': ['Marry me',\n",
"    'I love You',\n",
"    'date me',\n",
"    'chat with me',\n",
"    'I am bored'],\n",
"   'responses': ['Please ask organisation related query.'],\n",
"   'context': ['']},\n",
"  {'tag': 'noans',\n",
"   'patterns': ['why', 'how', 'when', 'I', 'you'],\n",
"   'responses': [\"Can't Understand Your Question\",\n",
"    'please elaborate your question'],\n",
"   'context': ['']},\n",
"  {'tag': 'turnover',\n",
"   'patterns': ['what is the turnover of the company at present?'],\n",
"   'responses': ['10 Million Ton'],\n",
"   'context': ['']},\n",
"  {'tag': 'predict_delay',\n",
"   'patterns': ['why is order 23A12 delayed'],\n",
"   'responses': ['delay is due to manufacuring',\n",
"    'Delay is due to inavailable components'],\n",
"   'context': ['']},\n",
"  {'tag': 'predict_delay',\n",
"   'patterns': ['why is order 23A12 delayed'],\n",
"   'responses': ['delay is due to manufacuring',\n",
"    'Delay is due to inavailable components'],\n",
"   'context': ['']},\n",
"  {'tag': 'name',\n",
"   'patterns': ['what is your name?', 'what should I call you?'],\n",
"   'responses': [\"I'm Bruno\", \"Hey!I'm Bruno\"],\n",
"   'context': ['']},\n",
"  {'tag': 'about',\n",
"   'patterns': ['how you doing?', 'how are you'],\n",
"   'responses': ['Thanks For Asking!How can I help you?'],\n",
"   'context': ['']},\n",
"  {'tag': 'configuration',\n",
"   'patterns': ['How to configure my laptop',\n",
"    'software configuration of laptop',\n",
"    'steps to configure laptop',\n",
"    'How to configure my computer',\n",
"    'software configuration of computer',\n",
"    'steps to configure computer',\n",
"    'How to configure my desktop',\n",
"    'software configuration of desktop',\n",
"    'steps to configure desktop'],\n",
"   'responses': ['search tab->control panel->select specific application->download update->give permission'],\n",
"   'context': ['']},\n",
"  {'tag': 'Weather',\n",
"   'patterns': ['what is the weather today?'],\n",
"   'responses': [\"It's 36C according to accuweather\"],\n",
"   'context': ['']},\n",
"  {'tag': 'leave',\n",
"   'patterns': ['Is Michel Sharma on leave?',\n",
"    'Is Siddhart Roy present today?',\n",
"    'Is Shantanu Bhatt in office?'],\n",
"   'responses': [\"Give me a moment!I'll just check\", 'yes', 'no'],\n",
"   'context': ['']},\n",
"  {'tag': 'hours',\n",
"   'patterns': ['Opening hours of the cafeteria?',\n",
"    'when does the cafeteria open',\n",
"    'office canteen opening time'],\n",
"   'responses': ['It is open from Monday-Saturday 9:00am-7:30pm'],\n",
"   'context': ['']},\n",
"  {'tag': 'cabin',\n",
```

```
        "   'patterns': ['where is VP cabin?',\n",
        "    'Where is Head IT cabin?',\n",
        "    'Where is AI department',\n",
        "    'Where is Project head department?',\n",
        "    'where is cafeteria',\n",
        "    'guide me to the canteen'],\n",
        "   'responses': ['Block-A 3rd floor 1st room',\n",
        "    'Block-B 2nd Floor',\n",
        "    'Block-D Ground Floor'],\n",
        "   'context': ['']},\n",
        "  {'tag': 'domain',\n",
        "   'patterns': ['How to improve team members domain knowledge',\n",
        "    'improving domain knowledge of team members'],\n",
        "   'responses': ['set up key meetings and workshop,create a shared drive for information,Hold informal sharing sess
        "   'context': ['']}]}"
       ]
      },
      "execution_count": 1,
      "metadata": {},
      "output_type": "execute_result"
     }
    ],
    "source": [
     "import json\n",
     "import random\n",
     "import string\n",
     "import nltk \n",
     "import numpy as np\n",
     "\n",
     "#reading json file\n",
     "file= open('intents.json').read()\n",
     "\n",
     "chats = json.loads(file)\n",
     "\n",
     "len(chats['intents'])\n",
     "chats\n",
     "\n"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 2,
    "id": "d7ebec06",
    "metadata": {},
    "outputs": [
     {
      "name": "stderr",
      "output_type": "stream",
      "text": [
       "[nltk_data] Downloading package punkt to\n",
       "[nltk_data]     C:\\Users\\la2022\\AppData\\Roaming\\nltk_data...\n",
       "[nltk_data]   Package punkt is already up-to-date!\n"
      ]
     }
    ],
    "source": [
     "# identifying features and target.\n",
     "# tokenizing words is achieved  using nltk.word_tokenize.\n",
     "# data_X represents features and data_Y represents tags of the patterns.\n",
     "#  words list contains all tokens , classes contains the corresponding tags, both are sorted und duplicate-cleaned.\n
     "#The first step here is to prepare our input data by identifying features represented by chats\n",
     "# and goals represented by tags. Additionally, we need to tokenize the chat texts with nltk.word_tokenize. \n",
     "#Some data cleansing is required to remove the tags and punctuation as these cannot provide meaningful information\n",
     "#to the model.\n",
     "nltk.download('punkt')\n",
     "from nltk.stem.lancaster import LancasterStemmer\n",
     "stemmer = LancasterStemmer()\n",
     "ignore = ['?']\n",
     "words =[]\n",
     "classes = []\n",
     "docs = []\n",
     "data_Y =[]\n",
     "\n",
     "for chat in chats['intents']:\n",
     "    for patt in chat['patterns']:\n",
     "        tokens = nltk.word_tokenize(patt)#tokenize the pattern\n",
     "        words.extend(tokens)##add tokens to words list\n",
     "        docs.append((tokens, chat['tag'])) ## append the pattern to\n",
     "        data_Y.append(chat['tag'])\n",
     "        \n",
     "    #add tags to classes list\n",
     "        if chat['tag'] not in classes:\n",
     "            classes.append(chat['tag'])"
    ]
   },
   {
    "cell_type": "code",
    "execution_count": 3,
    "id": "9cbc742b",
    "metadata": {},
    "outputs": [
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "121 documents\n",
       "43 classes ['HR_related_problem', 'Location', 'Weather', 'about', 'appointment status', 'cabin', 'check_leave', 'com
       "226 unique stemmed words [\"'s\", ',', '.', '23a12', '23a31', '32712', '345a23', '431b67', '561a24', '562b78', '@',
      ]
     },
     {
      "name": "stderr",
      "output_type": "stream",
      "text": [
```

```
      "[nltk_data] Downloading package punkt to\n",
      "[nltk_data]     C:\\Users\\la2022\\AppData\\Roaming\\nltk_data...\n",
      "[nltk_data]   Package punkt is already up-to-date!\n"
     ]
    }
   ],
   "source": [
    "#lemmatize all words (get the stems) and convert them to lower cases, remove punctuations\n",
    "import nltk\n",
    "nltk.download('punkt')\n",
    "from nltk.stem.lancaster import LancasterStemmer\n",
    "stemmer = LancasterStemmer()\n",
    "words = [stemmer.stem(w.lower()) for w in words if w not in ignore]\n",
    "words = sorted(list(set(words)))\n",
    "\n",
    "# remove duplicate classes\n",
    "classes = sorted(list(set(classes)))\n",
    "\n",
    "print (len(docs), \"documents\")\n",
    "print (len(classes), \"classes\", classes)\n",
    "print (len(words), \"unique stemmed words\", words)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 22,
   "id": "3e525b66",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "43\n"
     ]
    }
   ],
   "source": [
    "# create training data\n",
    "# bag-of-words is used to convert words into numbers. \n",
    "#  words and classes act as a vocabulary for patterns and tags respectively.\n",
    "# We?ll use words and classes to create two arrays, of the same lengths as the sizes of two vocabulary lists, for every\n",
    "# The array will have values 1 if the word is present in the patterns/tag of the chat, otherwise 0. \n",
    "\n",
    "training = []\n",
    "#output = []\n",
    "# create an empty array for output\n",
    "output_empty = [0] * len(classes)\n",
    "\n",
    "# create training set, bag of words for each sentence\n",
    "for doc in docs:\n",
    "    # initialize bag of words\n",
    "    bag = []\n",
    "    # list of tokenized words for the pattern\n",
    "    pattern_words = doc[0]\n",
    "    # stemming each word\n",
    "    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]\n",
    "    # create bag of words array\n",
    "    for w in words:\n",
    "        bag.append(1) if w in pattern_words else bag.append(0)\n",
    "\n",
    "    # output is '1' for current tag and '0' for rest of other tags\n",
    "    output_row = list(output_empty)\n",
    "    output_row[classes.index(doc[1])] = 1\n",
    "\n",
    "    training.append([bag, output_row])\n",
    "print(len(classes))\n",
    "# shuffling features and turning it into np.array\n",
    "random.shuffle(training)\n",
    "training = np.array(training, dtype=object)\n",
    "# creating training lists\n",
    "train_x = list(training[:,0])\n",
    "train_y = list(training[:,1])  "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 23,
   "id": "d20dbfa3",
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "code",
   "execution_count": 24,
   "id": "dde0fc51",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "226"
      ]
     },
     "execution_count": 24,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "len(train_x[0])\n",
    "len(words)"
   ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": 25,
   "id": "9869accd",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "<bound method Model.summary of <keras.engine.sequential.Sequential object at 0x00000242DB73BB20>>\n",
      "Epoch 1/1000\n",
      "1/1 [==============================] - 1s 505ms/step - loss: 3.7659 - accuracy: 0.0496\n",
      "Epoch 2/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 3.6834 - accuracy: 0.0992\n",
      "Epoch 3/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 3.6309 - accuracy: 0.0992\n",
      "Epoch 4/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 3.5270 - accuracy: 0.1488\n",
      "Epoch 5/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 3.3417 - accuracy: 0.1901\n",
      "Epoch 6/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 3.2504 - accuracy: 0.2314\n",
      "Epoch 7/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 3.1343 - accuracy: 0.2066\n",
      "Epoch 8/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 3.0527 - accuracy: 0.2727\n",
      "Epoch 9/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 2.8387 - accuracy: 0.2479\n",
      "Epoch 10/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 2.7696 - accuracy: 0.3058\n",
      "Epoch 11/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 2.5829 - accuracy: 0.2810\n",
      "Epoch 12/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 2.3277 - accuracy: 0.4050\n",
      "Epoch 13/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 2.3160 - accuracy: 0.3802\n",
      "Epoch 14/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 2.0059 - accuracy: 0.4711\n",
      "Epoch 15/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 2.0114 - accuracy: 0.4793\n",
      "Epoch 16/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 1.7427 - accuracy: 0.5455\n",
      "Epoch 17/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 1.5735 - accuracy: 0.6364\n",
      "Epoch 18/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 1.6602 - accuracy: 0.5537\n",
      "Epoch 19/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 1.3407 - accuracy: 0.6281\n",
      "Epoch 20/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 1.3626 - accuracy: 0.6612\n",
      "Epoch 21/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 1.3831 - accuracy: 0.6281\n",
      "Epoch 22/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 1.2886 - accuracy: 0.6281\n",
      "Epoch 23/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 1.0809 - accuracy: 0.7190\n",
      "Epoch 24/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.9891 - accuracy: 0.7438\n",
      "Epoch 25/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 1.1103 - accuracy: 0.6612\n",
      "Epoch 26/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.8419 - accuracy: 0.7355\n",
      "Epoch 27/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.8865 - accuracy: 0.7521\n",
      "Epoch 28/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.8370 - accuracy: 0.7603\n",
      "Epoch 29/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.6702 - accuracy: 0.8182\n",
      "Epoch 30/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.6559 - accuracy: 0.8182\n",
      "Epoch 31/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.6586 - accuracy: 0.7769\n",
      "Epoch 32/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.5750 - accuracy: 0.8347\n",
      "Epoch 33/1000\n",
      "1/1 [==============================] - 0s 6ms/step - loss: 0.5086 - accuracy: 0.9008\n",
      "Epoch 34/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.5515 - accuracy: 0.8512\n",
      "Epoch 35/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.5570 - accuracy: 0.8099\n",
      "Epoch 36/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.5105 - accuracy: 0.8760\n",
      "Epoch 37/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.5252 - accuracy: 0.8595\n",
      "Epoch 38/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.5075 - accuracy: 0.8512\n",
      "Epoch 39/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.4486 - accuracy: 0.8926\n",
      "Epoch 40/1000\n",
      "1/1 [==============================] - 0s 1ms/step - loss: 0.4110 - accuracy: 0.8760\n",
      "Epoch 41/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.4218 - accuracy: 0.8595\n",
      "Epoch 42/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.3931 - accuracy: 0.8843\n",
      "Epoch 43/1000\n",
      "1/1 [==============================] - 0s 1ms/step - loss: 0.3926 - accuracy: 0.8843\n",
      "Epoch 44/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.3959 - accuracy: 0.9174\n",
      "Epoch 45/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.4826 - accuracy: 0.8678\n",
      "Epoch 46/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2702 - accuracy: 0.9008\n",
```

```
      "Epoch 47/1000\n",
      "1/1 [==============================] - 0s 5ms/step - loss: 0.2810 - accuracy: 0.9339\n",
      "Epoch 48/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.3455 - accuracy: 0.9091\n",
      "Epoch 49/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.3541 - accuracy: 0.8595\n",
      "Epoch 50/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.3750 - accuracy: 0.8926\n",
      "Epoch 51/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.2268 - accuracy: 0.9421\n",
      "Epoch 52/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.3122 - accuracy: 0.9256\n",
      "Epoch 53/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.3055 - accuracy: 0.9091\n",
      "Epoch 54/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.2933 - accuracy: 0.9008\n",
      "Epoch 55/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.3300 - accuracy: 0.9174\n",
      "Epoch 56/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.3442 - accuracy: 0.8926\n",
      "Epoch 57/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.3737 - accuracy: 0.8926\n",
      "Epoch 58/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.3264 - accuracy: 0.9091\n",
      "Epoch 59/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.2197 - accuracy: 0.9339\n",
      "Epoch 60/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.3110 - accuracy: 0.9091\n",
      "Epoch 61/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2539 - accuracy: 0.9008\n",
      "Epoch 62/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2092 - accuracy: 0.9504\n",
      "Epoch 63/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2540 - accuracy: 0.9256\n",
      "Epoch 64/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2630 - accuracy: 0.9256\n",
      "Epoch 65/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1726 - accuracy: 0.9504\n",
      "Epoch 66/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.4095 - accuracy: 0.8430\n",
      "Epoch 67/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2767 - accuracy: 0.9174\n",
      "Epoch 68/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.2170 - accuracy: 0.9421\n",
      "Epoch 69/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1921 - accuracy: 0.9339\n",
      "Epoch 70/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1677 - accuracy: 0.9587\n",
      "Epoch 71/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.2192 - accuracy: 0.9256\n",
      "Epoch 72/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.2610 - accuracy: 0.9091\n",
      "Epoch 73/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1666 - accuracy: 0.9256\n",
      "Epoch 74/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1865 - accuracy: 0.9587\n",
      "Epoch 75/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1453 - accuracy: 0.9752\n",
      "Epoch 76/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.2214 - accuracy: 0.9421\n",
      "Epoch 77/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1736 - accuracy: 0.9504\n",
      "Epoch 78/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2181 - accuracy: 0.9421\n",
      "Epoch 79/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1410 - accuracy: 0.9587\n",
      "Epoch 80/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2698 - accuracy: 0.9174\n",
      "Epoch 81/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1984 - accuracy: 0.9339\n",
      "Epoch 82/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1496 - accuracy: 0.9339\n",
      "Epoch 83/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2061 - accuracy: 0.9339\n",
      "Epoch 84/1000\n",
      "1/1 [==============================] - 0s 5ms/step - loss: 0.2845 - accuracy: 0.9091\n",
      "Epoch 85/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.2963 - accuracy: 0.8926\n",
      "Epoch 86/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1441 - accuracy: 0.9587\n",
      "Epoch 87/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1565 - accuracy: 0.9504\n",
      "Epoch 88/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2321 - accuracy: 0.9091\n",
      "Epoch 89/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.2322 - accuracy: 0.9256\n",
      "Epoch 90/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1512 - accuracy: 0.9421\n",
      "Epoch 91/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2105 - accuracy: 0.9091\n",
      "Epoch 92/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.2363 - accuracy: 0.9256\n",
      "Epoch 93/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2359 - accuracy: 0.9339\n",
      "Epoch 94/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0902 - accuracy: 0.9752\n",
      "Epoch 95/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1692 - accuracy: 0.9587\n",
```

```
"Epoch 96/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1514 - accuracy: 0.9421\n",
"Epoch 97/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1503 - accuracy: 0.9504\n",
"Epoch 98/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.2015 - accuracy: 0.9421\n",
"Epoch 99/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0965 - accuracy: 0.9669\n",
"Epoch 100/1000\n",
"1/1 [==============================] - 0s 21ms/step - loss: 0.1542 - accuracy: 0.9421\n",
"Epoch 101/1000\n",
"1/1 [==============================] - 0s 20ms/step - loss: 0.1426 - accuracy: 0.9587\n",
"Epoch 102/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1262 - accuracy: 0.9669\n",
"Epoch 103/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.2002 - accuracy: 0.9587\n",
"Epoch 104/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0922 - accuracy: 0.9669\n",
"Epoch 105/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1227 - accuracy: 0.9669\n",
"Epoch 106/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1985 - accuracy: 0.9421\n",
"Epoch 107/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.2200 - accuracy: 0.9008\n",
"Epoch 108/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1298 - accuracy: 0.9587\n",
"Epoch 109/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.2551 - accuracy: 0.9174\n",
"Epoch 110/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1171 - accuracy: 0.9752\n",
"Epoch 111/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1350 - accuracy: 0.9504\n",
"Epoch 112/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1314 - accuracy: 0.9669\n",
"Epoch 113/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.2231 - accuracy: 0.9256\n",
"Epoch 114/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1504 - accuracy: 0.9587\n",
"Epoch 115/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.2298 - accuracy: 0.9256\n",
"Epoch 116/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1584 - accuracy: 0.9669\n",
"Epoch 117/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1921 - accuracy: 0.9421\n",
"Epoch 118/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1647 - accuracy: 0.9587\n",
"Epoch 119/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1406 - accuracy: 0.9587\n",
"Epoch 120/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.2276 - accuracy: 0.9504\n",
"Epoch 121/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1768 - accuracy: 0.9339\n",
"Epoch 122/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.0872 - accuracy: 0.9752\n",
"Epoch 123/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1783 - accuracy: 0.9339\n",
"Epoch 124/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1598 - accuracy: 0.9504\n",
"Epoch 125/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1896 - accuracy: 0.9504\n",
"Epoch 126/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1669 - accuracy: 0.9339\n",
"Epoch 127/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1182 - accuracy: 0.9669\n",
"Epoch 128/1000\n",
"1/1 [==============================] - 0s 20ms/step - loss: 0.1662 - accuracy: 0.9504\n",
"Epoch 129/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1778 - accuracy: 0.9174\n",
"Epoch 130/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0947 - accuracy: 0.9587\n",
"Epoch 131/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1544 - accuracy: 0.9339\n",
"Epoch 132/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1504 - accuracy: 0.9504\n",
"Epoch 133/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0870 - accuracy: 0.9669\n",
"Epoch 134/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1601 - accuracy: 0.9091\n",
"Epoch 135/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1086 - accuracy: 0.9835\n",
"Epoch 136/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1449 - accuracy: 0.9669\n",
"Epoch 137/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1202 - accuracy: 0.9504\n",
"Epoch 138/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1416 - accuracy: 0.9421\n",
"Epoch 139/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0910 - accuracy: 0.9504\n",
"Epoch 140/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.0992 - accuracy: 0.9587\n",
"Epoch 141/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1117 - accuracy: 0.9835\n",
"Epoch 142/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1221 - accuracy: 0.9421\n",
"Epoch 143/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.2217 - accuracy: 0.9504\n",
"Epoch 144/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1143 - accuracy: 0.9587\n",
"Epoch 145/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1455 - accuracy: 0.9504\n",
"Epoch 146/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1788 - accuracy: 0.9504\n",
"Epoch 147/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1346 - accuracy: 0.9421\n",
```

```
      "Epoch 148/1000\n",
      "1/1 [==============================] - 0s 1ms/step - loss: 0.2230 - accuracy: 0.9421\n",
      "Epoch 149/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1952 - accuracy: 0.9421\n",
      "Epoch 150/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1469 - accuracy: 0.9421\n",
      "Epoch 151/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0930 - accuracy: 0.9669\n",
      "Epoch 152/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1536 - accuracy: 0.9587\n",
      "Epoch 153/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1058 - accuracy: 0.9669\n",
      "Epoch 154/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1504 - accuracy: 0.9504\n",
      "Epoch 155/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1423 - accuracy: 0.9504\n",
      "Epoch 156/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0928 - accuracy: 0.9669\n",
      "Epoch 157/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1215 - accuracy: 0.9504\n",
      "Epoch 158/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1847 - accuracy: 0.9669\n",
      "Epoch 159/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1055 - accuracy: 0.9752\n",
      "Epoch 160/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1356 - accuracy: 0.9339\n",
      "Epoch 161/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1414 - accuracy: 0.9587\n",
      "Epoch 162/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1926 - accuracy: 0.9421\n",
      "Epoch 163/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1518 - accuracy: 0.9421\n",
      "Epoch 164/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1065 - accuracy: 0.9587\n",
      "Epoch 165/1000\n",
      "1/1 [==============================] - 0s 22ms/step - loss: 0.1213 - accuracy: 0.9587\n",
      "Epoch 166/1000\n",
      "1/1 [==============================] - 0s 23ms/step - loss: 0.1048 - accuracy: 0.9587\n",
      "Epoch 167/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1553 - accuracy: 0.9587\n",
      "Epoch 168/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1081 - accuracy: 0.9669\n",
      "Epoch 169/1000\n",
      "1/1 [==============================] - 0s 21ms/step - loss: 0.1091 - accuracy: 0.9835\n",
      "Epoch 170/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1457 - accuracy: 0.9421\n",
      "Epoch 171/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1445 - accuracy: 0.9504\n",
      "Epoch 172/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.1063 - accuracy: 0.9587\n",
      "Epoch 173/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0868 - accuracy: 0.9752\n",
      "Epoch 174/1000\n",
      "1/1 [==============================] - 0s 21ms/step - loss: 0.1035 - accuracy: 0.9835\n",
      "Epoch 175/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1464 - accuracy: 0.9421\n",
      "Epoch 176/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1541 - accuracy: 0.9504\n",
      "Epoch 177/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1223 - accuracy: 0.9504\n",
      "Epoch 178/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1459 - accuracy: 0.9504\n",
      "Epoch 179/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1278 - accuracy: 0.9752\n",
      "Epoch 180/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1066 - accuracy: 0.9669\n",
      "Epoch 181/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1141 - accuracy: 0.9669\n",
      "Epoch 182/1000\n",
      "1/1 [==============================] - 0s 21ms/step - loss: 0.1689 - accuracy: 0.9421\n",
      "Epoch 183/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1340 - accuracy: 0.9504\n",
      "Epoch 184/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0691 - accuracy: 0.9504\n",
      "Epoch 185/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1904 - accuracy: 0.9174\n",
      "Epoch 186/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1011 - accuracy: 0.9669\n",
      "Epoch 187/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.2476 - accuracy: 0.9421\n",
      "Epoch 188/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0413 - accuracy: 0.9835\n",
      "Epoch 189/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1335 - accuracy: 0.9669\n",
      "Epoch 190/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1552 - accuracy: 0.9256\n",
      "Epoch 191/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.0999 - accuracy: 0.9669\n",
      "Epoch 192/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1934 - accuracy: 0.9256\n",
      "Epoch 193/1000\n",
      "1/1 [==============================] - 0s 21ms/step - loss: 0.0877 - accuracy: 0.9669\n",
      "Epoch 194/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1142 - accuracy: 0.9421\n",
      "Epoch 195/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1082 - accuracy: 0.9669\n",
      "Epoch 196/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0610 - accuracy: 0.9835\n",
```

```
      "Epoch 197/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1230 - accuracy: 0.9669\n",
      "Epoch 198/1000\n",
      "1/1 [==============================] - 0s 3ms/step - loss: 0.1633 - accuracy: 0.9421\n",
      "Epoch 199/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0908 - accuracy: 0.9587\n",
      "Epoch 200/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1209 - accuracy: 0.9587\n",
      "Epoch 201/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.2120 - accuracy: 0.9421\n",
      "Epoch 202/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1103 - accuracy: 0.9669\n",
      "Epoch 203/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1158 - accuracy: 0.9587\n",
      "Epoch 204/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.0980 - accuracy: 0.9587\n",
      "Epoch 205/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1038 - accuracy: 0.9504\n",
      "Epoch 206/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1078 - accuracy: 0.9669\n",
      "Epoch 207/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0763 - accuracy: 0.9669\n",
      "Epoch 208/1000\n",
      "1/1 [==============================] - 0s 6ms/step - loss: 0.1924 - accuracy: 0.9339\n",
      "Epoch 209/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0716 - accuracy: 0.9669\n",
      "Epoch 210/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1395 - accuracy: 0.9421\n",
      "Epoch 211/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1115 - accuracy: 0.9669\n",
      "Epoch 212/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.0760 - accuracy: 0.9752\n",
      "Epoch 213/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0904 - accuracy: 0.9587\n",
      "Epoch 214/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1141 - accuracy: 0.9669\n",
      "Epoch 215/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1215 - accuracy: 0.9752\n",
      "Epoch 216/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1089 - accuracy: 0.9669\n",
      "Epoch 217/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1851 - accuracy: 0.9504\n",
      "Epoch 218/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0588 - accuracy: 0.9835\n",
      "Epoch 219/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1199 - accuracy: 0.9504\n",
      "Epoch 220/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1693 - accuracy: 0.9421\n",
      "Epoch 221/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1288 - accuracy: 0.9339\n",
      "Epoch 222/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1012 - accuracy: 0.9504\n",
      "Epoch 223/1000\n",
      "1/1 [==============================] - 0s 22ms/step - loss: 0.1227 - accuracy: 0.9587\n",
      "Epoch 224/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1285 - accuracy: 0.9504\n",
      "Epoch 225/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1178 - accuracy: 0.9669\n",
      "Epoch 226/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0824 - accuracy: 0.9752\n",
      "Epoch 227/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1340 - accuracy: 0.9587\n",
      "Epoch 228/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1507 - accuracy: 0.9587\n",
      "Epoch 229/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1198 - accuracy: 0.9669\n",
      "Epoch 230/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0719 - accuracy: 0.9669\n",
      "Epoch 231/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0586 - accuracy: 0.9835\n",
      "Epoch 232/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1535 - accuracy: 0.9504\n",
      "Epoch 233/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0789 - accuracy: 0.9669\n",
      "Epoch 234/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.0973 - accuracy: 0.9669\n",
      "Epoch 235/1000\n",
      "1/1 [==============================] - 0s 2ms/step - loss: 0.1935 - accuracy: 0.9256\n",
      "Epoch 236/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0708 - accuracy: 0.9669\n",
      "Epoch 237/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0813 - accuracy: 0.9669\n",
      "Epoch 238/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1971 - accuracy: 0.9504\n",
      "Epoch 239/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.2013 - accuracy: 0.9339\n",
      "Epoch 240/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1988 - accuracy: 0.9421\n",
      "Epoch 241/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1330 - accuracy: 0.9421\n",
      "Epoch 242/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0823 - accuracy: 0.9752\n",
      "Epoch 243/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.2727 - accuracy: 0.9008\n",
      "Epoch 244/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1094 - accuracy: 0.9504\n",
      "Epoch 245/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1337 - accuracy: 0.9752\n",
```

```
"Epoch 246/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0873 - accuracy: 0.9669\n",
"Epoch 247/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1123 - accuracy: 0.9587\n",
"Epoch 248/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1450 - accuracy: 0.9587\n",
"Epoch 249/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1119 - accuracy: 0.9504\n",
"Epoch 250/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1162 - accuracy: 0.9421\n",
"Epoch 251/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1065 - accuracy: 0.9669\n",
"Epoch 252/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1464 - accuracy: 0.9256\n",
"Epoch 253/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1611 - accuracy: 0.9504\n",
"Epoch 254/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1358 - accuracy: 0.9587\n",
"Epoch 255/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1469 - accuracy: 0.9421\n",
"Epoch 256/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0700 - accuracy: 0.9752\n",
"Epoch 257/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1002 - accuracy: 0.9669\n",
"Epoch 258/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1409 - accuracy: 0.9587\n",
"Epoch 259/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1332 - accuracy: 0.9587\n",
"Epoch 260/1000\n",
"1/1 [==============================] - 0s 20ms/step - loss: 0.1279 - accuracy: 0.9421\n",
"Epoch 261/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0609 - accuracy: 0.9835\n",
"Epoch 262/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0866 - accuracy: 0.9669\n",
"Epoch 263/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0981 - accuracy: 0.9587\n",
"Epoch 264/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1656 - accuracy: 0.9504\n",
"Epoch 265/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0709 - accuracy: 0.9752\n",
"Epoch 266/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0935 - accuracy: 0.9587\n",
"Epoch 267/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0996 - accuracy: 0.9752\n",
"Epoch 268/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1751 - accuracy: 0.9504\n",
"Epoch 269/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1524 - accuracy: 0.9339\n",
"Epoch 270/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1350 - accuracy: 0.9339\n",
"Epoch 271/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1615 - accuracy: 0.9339\n",
"Epoch 272/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1003 - accuracy: 0.9752\n",
"Epoch 273/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.0980 - accuracy: 0.9421\n",
"Epoch 274/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1360 - accuracy: 0.9587\n",
"Epoch 275/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1675 - accuracy: 0.9587\n",
"Epoch 276/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1459 - accuracy: 0.9339\n",
"Epoch 277/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.1905 - accuracy: 0.9339\n",
"Epoch 278/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0911 - accuracy: 0.9504\n",
"Epoch 279/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0730 - accuracy: 0.9752\n",
"Epoch 280/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1046 - accuracy: 0.9669\n",
"Epoch 281/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0867 - accuracy: 0.9752\n",
"Epoch 282/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0458 - accuracy: 0.9917\n",
"Epoch 283/1000\n",
"1/1 [==============================] - 0s 609us/step - loss: 0.0764 - accuracy: 0.9835\n",
"Epoch 284/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1108 - accuracy: 0.9587\n",
"Epoch 285/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.0823 - accuracy: 0.9835\n",
"Epoch 286/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1933 - accuracy: 0.9421\n",
"Epoch 287/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0732 - accuracy: 0.9587\n",
"Epoch 288/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0497 - accuracy: 0.9752\n",
"Epoch 289/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1299 - accuracy: 0.9587\n",
"Epoch 290/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0632 - accuracy: 0.9752\n",
"Epoch 291/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0859 - accuracy: 0.9669\n",
"Epoch 292/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1492 - accuracy: 0.9504\n",
"Epoch 293/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1856 - accuracy: 0.9339\n",
"Epoch 294/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0796 - accuracy: 0.9669\n",
"Epoch 295/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0413 - accuracy: 0.9917\n",
"Epoch 296/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0986 - accuracy: 0.9421\n",
"Epoch 297/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1783 - accuracy: 0.9421\n",
```

```
      "Epoch 298/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1439 - accuracy: 0.9587\n",
      "Epoch 299/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1058 - accuracy: 0.9669\n",
      "Epoch 300/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1209 - accuracy: 0.9669\n",
      "Epoch 301/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1645 - accuracy: 0.9669\n",
      "Epoch 302/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1186 - accuracy: 0.9587\n",
      "Epoch 303/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1153 - accuracy: 0.9587\n",
      "Epoch 304/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0951 - accuracy: 0.9587\n",
      "Epoch 305/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1280 - accuracy: 0.9339\n",
      "Epoch 306/1000\n",
      "1/1 [==============================] - 0s 3ms/step - loss: 0.0917 - accuracy: 0.9669\n",
      "Epoch 307/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0929 - accuracy: 0.9669\n",
      "Epoch 308/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0892 - accuracy: 0.9752\n",
      "Epoch 309/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1784 - accuracy: 0.9835\n",
      "Epoch 310/1000\n",
      "1/1 [==============================] - 0s 2ms/step - loss: 0.0871 - accuracy: 0.9587\n",
      "Epoch 311/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1200 - accuracy: 0.9587\n",
      "Epoch 312/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0794 - accuracy: 0.9587\n",
      "Epoch 313/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1257 - accuracy: 0.9504\n",
      "Epoch 314/1000\n",
      "1/1 [==============================] - 0s 5ms/step - loss: 0.0656 - accuracy: 0.9835\n",
      "Epoch 315/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1607 - accuracy: 0.9339\n",
      "Epoch 316/1000\n",
      "1/1 [==============================] - 0s 2ms/step - loss: 0.0643 - accuracy: 0.9917\n",
      "Epoch 317/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0831 - accuracy: 0.9587\n",
      "Epoch 318/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1045 - accuracy: 0.9504\n",
      "Epoch 319/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1652 - accuracy: 0.9587\n",
      "Epoch 320/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1161 - accuracy: 0.9504\n",
      "Epoch 321/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0628 - accuracy: 0.9752\n",
      "Epoch 322/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1033 - accuracy: 0.9752\n",
      "Epoch 323/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.0654 - accuracy: 0.9669\n",
      "Epoch 324/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0621 - accuracy: 0.9669\n",
      "Epoch 325/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0978 - accuracy: 0.9587\n",
      "Epoch 326/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0921 - accuracy: 0.9587\n",
      "Epoch 327/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0844 - accuracy: 0.9669\n",
      "Epoch 328/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1465 - accuracy: 0.9587\n",
      "Epoch 329/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0556 - accuracy: 0.9835\n",
      "Epoch 330/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1671 - accuracy: 0.9421\n",
      "Epoch 331/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1073 - accuracy: 0.9669\n",
      "Epoch 332/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1355 - accuracy: 0.9669\n",
      "Epoch 333/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1722 - accuracy: 0.9421\n",
      "Epoch 334/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1326 - accuracy: 0.9256\n",
      "Epoch 335/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1392 - accuracy: 0.9421\n",
      "Epoch 336/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0609 - accuracy: 0.9752\n",
      "Epoch 337/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0772 - accuracy: 0.9752\n",
      "Epoch 338/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1331 - accuracy: 0.9339\n",
      "Epoch 339/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2081 - accuracy: 0.9091\n",
      "Epoch 340/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0547 - accuracy: 0.9669\n",
      "Epoch 341/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.0459 - accuracy: 0.9752\n",
      "Epoch 342/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.1381 - accuracy: 0.9669\n",
      "Epoch 343/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1135 - accuracy: 0.9504\n",
      "Epoch 344/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1606 - accuracy: 0.9504\n",
      "Epoch 345/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0711 - accuracy: 0.9669\n",
      "Epoch 346/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0461 - accuracy: 0.9835\n",
```

```
"Epoch 347/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0363 - accuracy: 1.0000\n",
"Epoch 348/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0364 - accuracy: 0.9917\n",
"Epoch 349/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1047 - accuracy: 0.9504\n",
"Epoch 350/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1706 - accuracy: 0.9339\n",
"Epoch 351/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0928 - accuracy: 0.9504\n",
"Epoch 352/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1188 - accuracy: 0.9587\n",
"Epoch 353/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0532 - accuracy: 0.9835\n",
"Epoch 354/1000\n",
"1/1 [==============================] - 0s 21ms/step - loss: 0.0953 - accuracy: 0.9835\n",
"Epoch 355/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1446 - accuracy: 0.9421\n",
"Epoch 356/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1243 - accuracy: 0.9587\n",
"Epoch 357/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.2580 - accuracy: 0.9339\n",
"Epoch 358/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.0891 - accuracy: 0.9669\n",
"Epoch 359/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0890 - accuracy: 0.9669\n",
"Epoch 360/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1305 - accuracy: 0.9587\n",
"Epoch 361/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1160 - accuracy: 0.9669\n",
"Epoch 362/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0880 - accuracy: 0.9752\n",
"Epoch 363/1000\n",
"1/1 [==============================] - 0s 6ms/step - loss: 0.1154 - accuracy: 0.9421\n",
"Epoch 364/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0741 - accuracy: 0.9669\n",
"Epoch 365/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0580 - accuracy: 0.9752\n",
"Epoch 366/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1399 - accuracy: 0.9421\n",
"Epoch 367/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.0918 - accuracy: 0.9587\n",
"Epoch 368/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1794 - accuracy: 0.9587\n",
"Epoch 369/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0954 - accuracy: 0.9587\n",
"Epoch 370/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0495 - accuracy: 0.9835\n",
"Epoch 371/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1348 - accuracy: 0.9587\n",
"Epoch 372/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0598 - accuracy: 0.9752\n",
"Epoch 373/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0725 - accuracy: 0.9835\n",
"Epoch 374/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1580 - accuracy: 0.9752\n",
"Epoch 375/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0961 - accuracy: 0.9669\n",
"Epoch 376/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.1133 - accuracy: 0.9669\n",
"Epoch 377/1000\n",
"1/1 [==============================] - 0s 20ms/step - loss: 0.1064 - accuracy: 0.9504\n",
"Epoch 378/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.1722 - accuracy: 0.9421\n",
"Epoch 379/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0557 - accuracy: 0.9835\n",
"Epoch 380/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0837 - accuracy: 0.9752\n",
"Epoch 381/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1611 - accuracy: 0.9669\n",
"Epoch 382/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.0502 - accuracy: 0.9835\n",
"Epoch 383/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.0888 - accuracy: 0.9669\n",
"Epoch 384/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1549 - accuracy: 0.9587\n",
"Epoch 385/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0437 - accuracy: 0.9835\n",
"Epoch 386/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1032 - accuracy: 0.9504\n",
"Epoch 387/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0639 - accuracy: 0.9835\n",
"Epoch 388/1000\n",
"1/1 [==============================] - 0s 21ms/step - loss: 0.0707 - accuracy: 0.9669\n",
"Epoch 389/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.2784 - accuracy: 0.9339\n",
"Epoch 390/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0960 - accuracy: 0.9587\n",
"Epoch 391/1000\n",
"1/1 [==============================] - 0s 5ms/step - loss: 0.1013 - accuracy: 0.9587\n",
"Epoch 392/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.0850 - accuracy: 0.9752\n",
"Epoch 393/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1516 - accuracy: 0.9421\n",
"Epoch 394/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.0482 - accuracy: 0.9835\n",
"Epoch 395/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0797 - accuracy: 0.9669\n",
"Epoch 396/1000\n",
"1/1 [==============================] - 0s 12ms/step - loss: 0.0904 - accuracy: 0.9504\n",
"Epoch 397/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0937 - accuracy: 0.9669\n",
"Epoch 398/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1338 - accuracy: 0.9587\n",
```

```
      "Epoch 399/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1268 - accuracy: 0.9504\n",
      "Epoch 400/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.0797 - accuracy: 0.9669\n",
      "Epoch 401/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1005 - accuracy: 0.9504\n",
      "Epoch 402/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1532 - accuracy: 0.9752\n",
      "Epoch 403/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1232 - accuracy: 0.9587\n",
      "Epoch 404/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2143 - accuracy: 0.9504\n",
      "Epoch 405/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0710 - accuracy: 0.9669\n",
      "Epoch 406/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0814 - accuracy: 0.9917\n",
      "Epoch 407/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1162 - accuracy: 0.9421\n",
      "Epoch 408/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0932 - accuracy: 0.9587\n",
      "Epoch 409/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0634 - accuracy: 0.9835\n",
      "Epoch 410/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0474 - accuracy: 0.9835\n",
      "Epoch 411/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1546 - accuracy: 0.9504\n",
      "Epoch 412/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0920 - accuracy: 0.9504\n",
      "Epoch 413/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0921 - accuracy: 0.9669\n",
      "Epoch 414/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1146 - accuracy: 0.9504\n",
      "Epoch 415/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1133 - accuracy: 0.9669\n",
      "Epoch 416/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1205 - accuracy: 0.9587\n",
      "Epoch 417/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0434 - accuracy: 0.9917\n",
      "Epoch 418/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0718 - accuracy: 0.9752\n",
      "Epoch 419/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1506 - accuracy: 0.9339\n",
      "Epoch 420/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1307 - accuracy: 0.9669\n",
      "Epoch 421/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1106 - accuracy: 0.9669\n",
      "Epoch 422/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1219 - accuracy: 0.9752\n",
      "Epoch 423/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1035 - accuracy: 0.9669\n",
      "Epoch 424/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0556 - accuracy: 0.9835\n",
      "Epoch 425/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0844 - accuracy: 0.9917\n",
      "Epoch 426/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0385 - accuracy: 0.9835\n",
      "Epoch 427/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1294 - accuracy: 0.9339\n",
      "Epoch 428/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1741 - accuracy: 0.9504\n",
      "Epoch 429/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0371 - accuracy: 0.9752\n",
      "Epoch 430/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1512 - accuracy: 0.9504\n",
      "Epoch 431/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1327 - accuracy: 0.9421\n",
      "Epoch 432/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1111 - accuracy: 0.9587\n",
      "Epoch 433/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1014 - accuracy: 0.9587\n",
      "Epoch 434/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0523 - accuracy: 0.9752\n",
      "Epoch 435/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0454 - accuracy: 0.9835\n",
      "Epoch 436/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0937 - accuracy: 0.9587\n",
      "Epoch 437/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1238 - accuracy: 0.9669\n",
      "Epoch 438/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1974 - accuracy: 0.9504\n",
      "Epoch 439/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0757 - accuracy: 0.9752\n",
      "Epoch 440/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.0646 - accuracy: 0.9752\n",
      "Epoch 441/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.0768 - accuracy: 0.9752\n",
      "Epoch 442/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0724 - accuracy: 0.9752\n",
      "Epoch 443/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0935 - accuracy: 0.9752\n",
      "Epoch 444/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1116 - accuracy: 0.9587\n",
      "Epoch 445/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1709 - accuracy: 0.9669\n",
      "Epoch 446/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1059 - accuracy: 0.9669\n",
      "Epoch 447/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1206 - accuracy: 0.9421\n",
```

```
      "Epoch 448/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0755 - accuracy: 0.9752\n",
      "Epoch 449/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1210 - accuracy: 0.9752\n",
      "Epoch 450/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0947 - accuracy: 0.9669\n",
      "Epoch 451/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0978 - accuracy: 0.9587\n",
      "Epoch 452/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0954 - accuracy: 0.9669\n",
      "Epoch 453/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1093 - accuracy: 0.9752\n",
      "Epoch 454/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0389 - accuracy: 0.9835\n",
      "Epoch 455/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0740 - accuracy: 0.9504\n",
      "Epoch 456/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0652 - accuracy: 0.9835\n",
      "Epoch 457/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1757 - accuracy: 0.9339\n",
      "Epoch 458/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0842 - accuracy: 0.9669\n",
      "Epoch 459/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1400 - accuracy: 0.9421\n",
      "Epoch 460/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0956 - accuracy: 0.9752\n",
      "Epoch 461/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1384 - accuracy: 0.9504\n",
      "Epoch 462/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0755 - accuracy: 0.9752\n",
      "Epoch 463/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1368 - accuracy: 0.9421\n",
      "Epoch 464/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1388 - accuracy: 0.9504\n",
      "Epoch 465/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0892 - accuracy: 0.9669\n",
      "Epoch 466/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1049 - accuracy: 0.9752\n",
      "Epoch 467/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1322 - accuracy: 0.9669\n",
      "Epoch 468/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1025 - accuracy: 0.9752\n",
      "Epoch 469/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1333 - accuracy: 0.9587\n",
      "Epoch 470/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.0759 - accuracy: 0.9669\n",
      "Epoch 471/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.0316 - accuracy: 1.0000\n",
      "Epoch 472/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0401 - accuracy: 0.9835\n",
      "Epoch 473/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.0862 - accuracy: 0.9669\n",
      "Epoch 474/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0676 - accuracy: 0.9752\n",
      "Epoch 475/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1988 - accuracy: 0.9587\n",
      "Epoch 476/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.0583 - accuracy: 0.9917\n",
      "Epoch 477/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0545 - accuracy: 0.9752\n",
      "Epoch 478/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1973 - accuracy: 0.9421\n",
      "Epoch 479/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0333 - accuracy: 0.9917\n",
      "Epoch 480/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1063 - accuracy: 0.9504\n",
      "Epoch 481/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1149 - accuracy: 0.9587\n",
      "Epoch 482/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0756 - accuracy: 0.9587\n",
      "Epoch 483/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0735 - accuracy: 0.9587\n",
      "Epoch 484/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1207 - accuracy: 0.9669\n",
      "Epoch 485/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1689 - accuracy: 0.9339\n",
      "Epoch 486/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0659 - accuracy: 0.9835\n",
      "Epoch 487/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0816 - accuracy: 0.9669\n",
      "Epoch 488/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0508 - accuracy: 0.9835\n",
      "Epoch 489/1000\n",
      "1/1 [==============================] - 0s 22ms/step - loss: 0.1094 - accuracy: 0.9917\n",
      "Epoch 490/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1460 - accuracy: 0.9504\n",
      "Epoch 491/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1732 - accuracy: 0.9504\n",
      "Epoch 492/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0754 - accuracy: 0.9752\n",
      "Epoch 493/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0625 - accuracy: 0.9752\n",
      "Epoch 494/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2012 - accuracy: 0.9504\n",
      "Epoch 495/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0764 - accuracy: 0.9835\n",
      "Epoch 496/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1579 - accuracy: 0.9587\n",
```

```
"Epoch 497/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1024 - accuracy: 0.9669\n",
"Epoch 498/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1585 - accuracy: 0.9421\n",
"Epoch 499/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1196 - accuracy: 0.9752\n",
"Epoch 500/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1283 - accuracy: 0.9504\n",
"Epoch 501/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.0493 - accuracy: 0.9669\n",
"Epoch 502/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0813 - accuracy: 0.9752\n",
"Epoch 503/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1629 - accuracy: 0.9587\n",
"Epoch 504/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0792 - accuracy: 0.9669\n",
"Epoch 505/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0875 - accuracy: 0.9669\n",
"Epoch 506/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1102 - accuracy: 0.9504\n",
"Epoch 507/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0889 - accuracy: 0.9587\n",
"Epoch 508/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0666 - accuracy: 0.9587\n",
"Epoch 509/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1221 - accuracy: 0.9504\n",
"Epoch 510/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1731 - accuracy: 0.9504\n",
"Epoch 511/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1635 - accuracy: 0.9587\n",
"Epoch 512/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1536 - accuracy: 0.9504\n",
"Epoch 513/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0929 - accuracy: 0.9669\n",
"Epoch 514/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0724 - accuracy: 0.9587\n",
"Epoch 515/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.0784 - accuracy: 0.9587\n",
"Epoch 516/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0597 - accuracy: 0.9835\n",
"Epoch 517/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0676 - accuracy: 0.9669\n",
"Epoch 518/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1593 - accuracy: 0.9504\n",
"Epoch 519/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0187 - accuracy: 1.0000\n",
"Epoch 520/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1865 - accuracy: 0.9421\n",
"Epoch 521/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1092 - accuracy: 0.9504\n",
"Epoch 522/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.2562 - accuracy: 0.9421\n",
"Epoch 523/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1160 - accuracy: 0.9587\n",
"Epoch 524/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.0727 - accuracy: 0.9752\n",
"Epoch 525/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1022 - accuracy: 0.9669\n",
"Epoch 526/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1374 - accuracy: 0.9504\n",
"Epoch 527/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1742 - accuracy: 0.9504\n",
"Epoch 528/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1683 - accuracy: 0.9504\n",
"Epoch 529/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1426 - accuracy: 0.9339\n",
"Epoch 530/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0514 - accuracy: 0.9835\n",
"Epoch 531/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.0564 - accuracy: 0.9752\n",
"Epoch 532/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0657 - accuracy: 0.9669\n",
"Epoch 533/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1233 - accuracy: 0.9421\n",
"Epoch 534/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1838 - accuracy: 0.9504\n",
"Epoch 535/1000\n",
"1/1 [==============================] - 0s 12ms/step - loss: 0.1562 - accuracy: 0.9339\n",
"Epoch 536/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0935 - accuracy: 0.9504\n",
"Epoch 537/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.0614 - accuracy: 0.9835\n",
"Epoch 538/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.0780 - accuracy: 0.9587\n",
"Epoch 539/1000\n",
"1/1 [==============================] - 0s 620us/step - loss: 0.0830 - accuracy: 0.9835\n",
"Epoch 540/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1762 - accuracy: 0.9587\n",
"Epoch 541/1000\n",
"1/1 [==============================] - 0s 2ms/step - loss: 0.0426 - accuracy: 0.9835\n",
"Epoch 542/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.2339 - accuracy: 0.9339\n",
"Epoch 543/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.0980 - accuracy: 0.9587\n",
"Epoch 544/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1123 - accuracy: 0.9504\n",
"Epoch 545/1000\n",
"1/1 [==============================] - 0s 1ms/step - loss: 0.2189 - accuracy: 0.9174\n",
"Epoch 546/1000\n",
"1/1 [==============================] - 0s 12ms/step - loss: 0.1028 - accuracy: 0.9752\n",
"Epoch 547/1000\n",
"1/1 [==============================] - 0s 3ms/step - loss: 0.0898 - accuracy: 0.9587\n",
"Epoch 548/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0464 - accuracy: 0.9835\n",
```

```
      "Epoch 549/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0548 - accuracy: 0.9752\n",
      "Epoch 550/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1207 - accuracy: 0.9587\n",
      "Epoch 551/1000\n",
      "1/1 [==============================] - 0s 3ms/step - loss: 0.1044 - accuracy: 0.9587\n",
      "Epoch 552/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0907 - accuracy: 0.9669\n",
      "Epoch 553/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0906 - accuracy: 0.9752\n",
      "Epoch 554/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1622 - accuracy: 0.9174\n",
      "Epoch 555/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1142 - accuracy: 0.9669\n",
      "Epoch 556/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1112 - accuracy: 0.9587\n",
      "Epoch 557/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1420 - accuracy: 0.9669\n",
      "Epoch 558/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0479 - accuracy: 0.9835\n",
      "Epoch 559/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1406 - accuracy: 0.9587\n",
      "Epoch 560/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1191 - accuracy: 0.9669\n",
      "Epoch 561/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.2214 - accuracy: 0.9339\n",
      "Epoch 562/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.3541 - accuracy: 0.9339\n",
      "Epoch 563/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0396 - accuracy: 0.9835\n",
      "Epoch 564/1000\n",
      "1/1 [==============================] - 0s 3ms/step - loss: 0.1054 - accuracy: 0.9752\n",
      "Epoch 565/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1448 - accuracy: 0.9669\n",
      "Epoch 566/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1968 - accuracy: 0.9339\n",
      "Epoch 567/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0968 - accuracy: 0.9752\n",
      "Epoch 568/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1401 - accuracy: 0.9669\n",
      "Epoch 569/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0659 - accuracy: 0.9752\n",
      "Epoch 570/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2137 - accuracy: 0.9174\n",
      "Epoch 571/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0740 - accuracy: 0.9669\n",
      "Epoch 572/1000\n",
      "1/1 [==============================] - 0s 21ms/step - loss: 0.0899 - accuracy: 0.9669\n",
      "Epoch 573/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0613 - accuracy: 0.9835\n",
      "Epoch 574/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1157 - accuracy: 0.9587\n",
      "Epoch 575/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.1140 - accuracy: 0.9752\n",
      "Epoch 576/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1228 - accuracy: 0.9504\n",
      "Epoch 577/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1624 - accuracy: 0.9421\n",
      "Epoch 578/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0556 - accuracy: 0.9752\n",
      "Epoch 579/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1126 - accuracy: 0.9587\n",
      "Epoch 580/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0812 - accuracy: 0.9752\n",
      "Epoch 581/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0813 - accuracy: 0.9752\n",
      "Epoch 582/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1203 - accuracy: 0.9504\n",
      "Epoch 583/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0637 - accuracy: 0.9669\n",
      "Epoch 584/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1007 - accuracy: 0.9835\n",
      "Epoch 585/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0506 - accuracy: 0.9752\n",
      "Epoch 586/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0927 - accuracy: 0.9587\n",
      "Epoch 587/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1334 - accuracy: 0.9504\n",
      "Epoch 588/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1667 - accuracy: 0.9421\n",
      "Epoch 589/1000\n",
      "1/1 [==============================] - 0s 3ms/step - loss: 0.0972 - accuracy: 0.9587\n",
      "Epoch 590/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0871 - accuracy: 0.9752\n",
      "Epoch 591/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1275 - accuracy: 0.9752\n",
      "Epoch 592/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0572 - accuracy: 0.9669\n",
      "Epoch 593/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1350 - accuracy: 0.9339\n",
      "Epoch 594/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1286 - accuracy: 0.9752\n",
      "Epoch 595/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1345 - accuracy: 0.9669\n",
      "Epoch 596/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0980 - accuracy: 0.9504\n",
      "Epoch 597/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0968 - accuracy: 0.9587\n",
```

```
 "Epoch 598/1000\n",
 "1/1 [==============================] - 0s 18ms/step - loss: 0.1374 - accuracy: 0.9587\n",
 "Epoch 599/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.0644 - accuracy: 0.9917\n",
 "Epoch 600/1000\n",
 "1/1 [==============================] - 0s 17ms/step - loss: 0.1377 - accuracy: 0.9504\n",
 "Epoch 601/1000\n",
 "1/1 [==============================] - 0s 10ms/step - loss: 0.1784 - accuracy: 0.9421\n",
 "Epoch 602/1000\n",
 "1/1 [==============================] - 0s 19ms/step - loss: 0.1192 - accuracy: 0.9752\n",
 "Epoch 603/1000\n",
 "1/1 [==============================] - 0s 11ms/step - loss: 0.0976 - accuracy: 0.9587\n",
 "Epoch 604/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.0630 - accuracy: 0.9835\n",
 "Epoch 605/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.3037 - accuracy: 0.9421\n",
 "Epoch 606/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.0534 - accuracy: 0.9752\n",
 "Epoch 607/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.1037 - accuracy: 0.9669\n",
 "Epoch 608/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.0779 - accuracy: 0.9669\n",
 "Epoch 609/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.0978 - accuracy: 0.9752\n",
 "Epoch 610/1000\n",
 "1/1 [==============================] - 0s 15ms/step - loss: 0.0860 - accuracy: 0.9669\n",
 "Epoch 611/1000\n",
 "1/1 [==============================] - 0s 16ms/step - loss: 0.0539 - accuracy: 0.9752\n",
 "Epoch 612/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.1551 - accuracy: 0.9504\n",
 "Epoch 613/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.2837 - accuracy: 0.9256\n",
 "Epoch 614/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.1561 - accuracy: 0.9504\n",
 "Epoch 615/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.0532 - accuracy: 0.9752\n",
 "Epoch 616/1000\n",
 "1/1 [==============================] - 0s 6ms/step - loss: 0.1387 - accuracy: 0.9504\n",
 "Epoch 617/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.0555 - accuracy: 0.9587\n",
 "Epoch 618/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.1043 - accuracy: 0.9587\n",
 "Epoch 619/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.0822 - accuracy: 0.9669\n",
 "Epoch 620/1000\n",
 "1/1 [==============================] - 0s 16ms/step - loss: 0.1036 - accuracy: 0.9752\n",
 "Epoch 621/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.1339 - accuracy: 0.9587\n",
 "Epoch 622/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.1999 - accuracy: 0.9669\n",
 "Epoch 623/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.0641 - accuracy: 0.9752\n",
 "Epoch 624/1000\n",
 "1/1 [==============================] - 0s 15ms/step - loss: 0.1883 - accuracy: 0.9339\n",
 "Epoch 625/1000\n",
 "1/1 [==============================] - 0s 17ms/step - loss: 0.2020 - accuracy: 0.9421\n",
 "Epoch 626/1000\n",
 "1/1 [==============================] - 0s 0s/step - loss: 0.1094 - accuracy: 0.9587\n",
 "Epoch 627/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.1725 - accuracy: 0.9504\n",
 "Epoch 628/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.1753 - accuracy: 0.9669\n",
 "Epoch 629/1000\n",
 "1/1 [==============================] - 0s 19ms/step - loss: 0.0994 - accuracy: 0.9669\n",
 "Epoch 630/1000\n",
 "1/1 [==============================] - 0s 9ms/step - loss: 0.0943 - accuracy: 0.9504\n",
 "Epoch 631/1000\n",
 "1/1 [==============================] - 0s 19ms/step - loss: 0.1392 - accuracy: 0.9587\n",
 "Epoch 632/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.1788 - accuracy: 0.9504\n",
 "Epoch 633/1000\n",
 "1/1 [==============================] - 0s 7ms/step - loss: 0.3051 - accuracy: 0.9504\n",
 "Epoch 634/1000\n",
 "1/1 [==============================] - 0s 15ms/step - loss: 0.0572 - accuracy: 0.9835\n",
 "Epoch 635/1000\n",
 "1/1 [==============================] - 0s 15ms/step - loss: 0.0328 - accuracy: 0.9835\n",
 "Epoch 636/1000\n",
 "1/1 [==============================] - 0s 11ms/step - loss: 0.0910 - accuracy: 0.9504\n",
 "Epoch 637/1000\n",
 "1/1 [==============================] - 0s 0s/step - loss: 0.1794 - accuracy: 0.9339\n",
 "Epoch 638/1000\n",
 "1/1 [==============================] - 0s 16ms/step - loss: 0.1204 - accuracy: 0.9587\n",
 "Epoch 639/1000\n",
 "1/1 [==============================] - 0s 14ms/step - loss: 0.2978 - accuracy: 0.9174\n",
 "Epoch 640/1000\n",
 "1/1 [==============================] - 0s 456us/step - loss: 0.1270 - accuracy: 0.9752\n",
 "Epoch 641/1000\n",
 "1/1 [==============================] - 0s 16ms/step - loss: 0.2088 - accuracy: 0.9669\n",
 "Epoch 642/1000\n",
 "1/1 [==============================] - 0s 10ms/step - loss: 0.0981 - accuracy: 0.9504\n",
 "Epoch 643/1000\n",
 "1/1 [==============================] - 0s 6ms/step - loss: 0.0383 - accuracy: 0.9835\n",
 "Epoch 644/1000\n",
 "1/1 [==============================] - 0s 0s/step - loss: 0.0606 - accuracy: 0.9835\n",
 "Epoch 645/1000\n",
 "1/1 [==============================] - 0s 11ms/step - loss: 0.1415 - accuracy: 0.9504\n",
 "Epoch 646/1000\n",
 "1/1 [==============================] - 0s 16ms/step - loss: 0.0537 - accuracy: 0.9752\n",
 "Epoch 647/1000\n",
 "1/1 [==============================] - 0s 0s/step - loss: 0.0925 - accuracy: 0.9587\n",
 "Epoch 648/1000\n",
 "1/1 [==============================] - 0s 8ms/step - loss: 0.0569 - accuracy: 0.9835\n",
 "Epoch 649/1000\n"
]
```

```
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0415 - accuracy: 0.9752\n",
      "Epoch 650/1000\n",
      "1/1 [==============================] - 0s 22ms/step - loss: 0.0569 - accuracy: 0.9835\n",
      "Epoch 651/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1449 - accuracy: 0.9587\n",
      "Epoch 652/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.2563 - accuracy: 0.9091\n",
      "Epoch 653/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2300 - accuracy: 0.9587\n",
      "Epoch 654/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.2021 - accuracy: 0.9339\n",
      "Epoch 655/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0614 - accuracy: 0.9752\n",
      "Epoch 656/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1277 - accuracy: 0.9669\n",
      "Epoch 657/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.0477 - accuracy: 0.9752\n",
      "Epoch 658/1000\n",
      "1/1 [==============================] - 0s 21ms/step - loss: 0.2144 - accuracy: 0.9504\n",
      "Epoch 659/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0796 - accuracy: 0.9504\n",
      "Epoch 660/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1023 - accuracy: 0.9669\n",
      "Epoch 661/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0763 - accuracy: 0.9752\n",
      "Epoch 662/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1220 - accuracy: 0.9752\n",
      "Epoch 663/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0906 - accuracy: 0.9669\n",
      "Epoch 664/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1289 - accuracy: 0.9339\n",
      "Epoch 665/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0866 - accuracy: 0.9669\n",
      "Epoch 666/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1080 - accuracy: 0.9587\n",
      "Epoch 667/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0938 - accuracy: 0.9669\n",
      "Epoch 668/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.2423 - accuracy: 0.9504\n",
      "Epoch 669/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0954 - accuracy: 0.9752\n",
      "Epoch 670/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.0431 - accuracy: 0.9835\n",
      "Epoch 671/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1778 - accuracy: 0.9587\n",
      "Epoch 672/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1156 - accuracy: 0.9421\n",
      "Epoch 673/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0774 - accuracy: 0.9504\n",
      "Epoch 674/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0480 - accuracy: 0.9835\n",
      "Epoch 675/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.2387 - accuracy: 0.9587\n",
      "Epoch 676/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2333 - accuracy: 0.9256\n",
      "Epoch 677/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1311 - accuracy: 0.9504\n",
      "Epoch 678/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0993 - accuracy: 0.9587\n",
      "Epoch 679/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1043 - accuracy: 0.9587\n",
      "Epoch 680/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1274 - accuracy: 0.9587\n",
      "Epoch 681/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1008 - accuracy: 0.9669\n",
      "Epoch 682/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1528 - accuracy: 0.9339\n",
      "Epoch 683/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0353 - accuracy: 0.9835\n",
      "Epoch 684/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1172 - accuracy: 0.9504\n",
      "Epoch 685/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1552 - accuracy: 0.9339\n",
      "Epoch 686/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.0343 - accuracy: 0.9917\n",
      "Epoch 687/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1793 - accuracy: 0.9587\n",
      "Epoch 688/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0853 - accuracy: 0.9752\n",
      "Epoch 689/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1881 - accuracy: 0.9504\n",
      "Epoch 690/1000\n",
      "1/1 [==============================] - 0s 22ms/step - loss: 0.0406 - accuracy: 0.9835\n",
      "Epoch 691/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.1639 - accuracy: 0.9669\n",
      "Epoch 692/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.2129 - accuracy: 0.9339\n",
      "Epoch 693/1000\n",
      "1/1 [==============================] - 0s 5ms/step - loss: 0.0809 - accuracy: 0.9669\n",
      "Epoch 694/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2233 - accuracy: 0.9174\n",
      "Epoch 695/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0836 - accuracy: 0.9669\n",
      "Epoch 696/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2104 - accuracy: 0.9504\n",
      "Epoch 697/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0500 - accuracy: 0.9752\n",
      "Epoch 698/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1613 - accuracy: 0.9256\n",
```

```
      "Epoch 699/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.0814 - accuracy: 0.9669\n",
      "Epoch 700/1000\n",
      "1/1 [==============================] - 0s 6ms/step - loss: 0.2061 - accuracy: 0.9504\n",
      "Epoch 701/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1998 - accuracy: 0.9587\n",
      "Epoch 702/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0709 - accuracy: 0.9669\n",
      "Epoch 703/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.2081 - accuracy: 0.9174\n",
      "Epoch 704/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1923 - accuracy: 0.9091\n",
      "Epoch 705/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0939 - accuracy: 0.9669\n",
      "Epoch 706/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2353 - accuracy: 0.9421\n",
      "Epoch 707/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1452 - accuracy: 0.9587\n",
      "Epoch 708/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1809 - accuracy: 0.9587\n",
      "Epoch 709/1000\n",
      "1/1 [==============================] - 0s 5ms/step - loss: 0.1460 - accuracy: 0.9587\n",
      "Epoch 710/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1069 - accuracy: 0.9669\n",
      "Epoch 711/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1084 - accuracy: 0.9587\n",
      "Epoch 712/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0988 - accuracy: 0.9587\n",
      "Epoch 713/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1846 - accuracy: 0.9421\n",
      "Epoch 714/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1027 - accuracy: 0.9752\n",
      "Epoch 715/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1048 - accuracy: 0.9669\n",
      "Epoch 716/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1107 - accuracy: 0.9421\n",
      "Epoch 717/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.2066 - accuracy: 0.9339\n",
      "Epoch 718/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1357 - accuracy: 0.9587\n",
      "Epoch 719/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0718 - accuracy: 0.9752\n",
      "Epoch 720/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0984 - accuracy: 0.9587\n",
      "Epoch 721/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0779 - accuracy: 0.9587\n",
      "Epoch 722/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0499 - accuracy: 0.9669\n",
      "Epoch 723/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1521 - accuracy: 0.9504\n",
      "Epoch 724/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0930 - accuracy: 0.9669\n",
      "Epoch 725/1000\n",
      "1/1 [==============================] - 0s 1ms/step - loss: 0.0601 - accuracy: 0.9669\n",
      "Epoch 726/1000\n",
      "1/1 [==============================] - 0s 4ms/step - loss: 0.0523 - accuracy: 0.9752\n",
      "Epoch 727/1000\n",
      "1/1 [==============================] - 0s 5ms/step - loss: 0.0484 - accuracy: 0.9752\n",
      "Epoch 728/1000\n",
      "1/1 [==============================] - 0s 0s/step - loss: 0.0690 - accuracy: 0.9669\n",
      "Epoch 729/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.2276 - accuracy: 0.9421\n",
      "Epoch 730/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 11ms/step - loss: 0.0809 - accuracy: 0.9835\n",
      "Epoch 731/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0414 - accuracy: 0.9835\n",
      "Epoch 732/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.2658 - accuracy: 0.9339\n",
      "Epoch 733/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0688 - accuracy: 0.9504\n",
      "Epoch 734/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.3197 - accuracy: 0.9256\n",
      "Epoch 735/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1210 - accuracy: 0.9587\n",
      "Epoch 736/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.1186 - accuracy: 0.9669\n",
      "Epoch 737/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1525 - accuracy: 0.9587\n",
      "Epoch 738/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1286 - accuracy: 0.9752\n",
      "Epoch 739/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1380 - accuracy: 0.9669\n",
      "Epoch 740/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1097 - accuracy: 0.9504\n",
      "Epoch 741/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.3302 - accuracy: 0.9008\n",
      "Epoch 742/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.0787 - accuracy: 0.9504\n",
      "Epoch 743/1000\n",
      "1/1 [==============================] - 0s 2ms/step - loss: 0.1484 - accuracy: 0.9752\n",
      "Epoch 744/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1889 - accuracy: 0.9504\n",
      "Epoch 745/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0922 - accuracy: 0.9669\n",
      "Epoch 746/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.0366 - accuracy: 0.9835\n",
      "Epoch 747/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2033 - accuracy: 0.9421\n",
```

```
"Epoch 748/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.0339 - accuracy: 0.9835\n",
"Epoch 749/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1623 - accuracy: 0.9504\n",
"Epoch 750/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.2442 - accuracy: 0.9339\n",
"Epoch 751/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1646 - accuracy: 0.9752\n",
"Epoch 752/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1087 - accuracy: 0.9752\n",
"Epoch 753/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1661 - accuracy: 0.9421\n",
"Epoch 754/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0285 - accuracy: 0.9917\n",
"Epoch 755/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1512 - accuracy: 0.9504\n",
"Epoch 756/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1181 - accuracy: 0.9587\n",
"Epoch 757/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1413 - accuracy: 0.9587\n",
"Epoch 758/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1351 - accuracy: 0.9587\n",
"Epoch 759/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.0545 - accuracy: 0.9835\n",
"Epoch 760/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1295 - accuracy: 0.9421\n",
"Epoch 761/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0705 - accuracy: 0.9669\n",
"Epoch 762/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0807 - accuracy: 0.9669\n",
"Epoch 763/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1070 - accuracy: 0.9587\n",
"Epoch 764/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0994 - accuracy: 0.9587\n",
"Epoch 765/1000\n",
"1/1 [==============================] - 0s 836us/step - loss: 0.1899 - accuracy: 0.9504\n",
"Epoch 766/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.0590 - accuracy: 0.9835\n",
"Epoch 767/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1157 - accuracy: 0.9752\n",
"Epoch 768/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.2498 - accuracy: 0.9504\n",
"Epoch 769/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.1116 - accuracy: 0.9752\n",
"Epoch 770/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0586 - accuracy: 0.9752\n",
"Epoch 771/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0591 - accuracy: 0.9669\n",
"Epoch 772/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0634 - accuracy: 0.9669\n",
"Epoch 773/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0944 - accuracy: 0.9669\n",
"Epoch 774/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.0719 - accuracy: 0.9669\n",
"Epoch 775/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.0665 - accuracy: 0.9669\n",
"Epoch 776/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1880 - accuracy: 0.9421\n",
"Epoch 777/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.2434 - accuracy: 0.9504\n",
"Epoch 778/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0550 - accuracy: 0.9669\n",
"Epoch 779/1000\n",
"1/1 [==============================] - 0s 4ms/step - loss: 0.1233 - accuracy: 0.9587\n",
"Epoch 780/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1327 - accuracy: 0.9669\n",
"Epoch 781/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0492 - accuracy: 0.9835\n",
"Epoch 782/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0526 - accuracy: 0.9669\n",
"Epoch 783/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.2167 - accuracy: 0.9421\n",
"Epoch 784/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.2185 - accuracy: 0.9174\n",
"Epoch 785/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.0610 - accuracy: 0.9587\n",
"Epoch 786/1000\n",
"1/1 [==============================] - 0s 20ms/step - loss: 0.0809 - accuracy: 0.9752\n",
"Epoch 787/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1520 - accuracy: 0.9587\n",
"Epoch 788/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1507 - accuracy: 0.9669\n",
"Epoch 789/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1069 - accuracy: 0.9752\n",
"Epoch 790/1000\n",
"1/1 [==============================] - 0s 2ms/step - loss: 0.2295 - accuracy: 0.9256\n",
"Epoch 791/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0960 - accuracy: 0.9752\n",
"Epoch 792/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1357 - accuracy: 0.9421\n",
"Epoch 793/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0924 - accuracy: 0.9669\n",
"Epoch 794/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.2033 - accuracy: 0.9421\n",
"Epoch 795/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1725 - accuracy: 0.9504\n",
"Epoch 796/1000\n",
"1/1 [==============================] - 0s 18ms/step - loss: 0.1511 - accuracy: 0.9587\n",
"Epoch 797/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1599 - accuracy: 0.9587\n",
"Epoch 798/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0853 - accuracy: 0.9587\n",
"Epoch 799/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1515 - accuracy: 0.9752\n",
```

```
      "Epoch 800/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.1892 - accuracy: 0.9421\n",
      "Epoch 801/1000\n",
      "1/1 [==============================] - 0s 11ms/step - loss: 0.2018 - accuracy: 0.9669\n",
      "Epoch 802/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.0645 - accuracy: 0.9669\n",
      "Epoch 803/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0563 - accuracy: 0.9835\n",
      "Epoch 804/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.1534 - accuracy: 0.9587\n",
      "Epoch 805/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1728 - accuracy: 0.9421\n",
      "Epoch 806/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1148 - accuracy: 0.9421\n",
      "Epoch 807/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1240 - accuracy: 0.9587\n",
      "Epoch 808/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1286 - accuracy: 0.9504\n",
      "Epoch 809/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.1576 - accuracy: 0.9339\n",
      "Epoch 810/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.0609 - accuracy: 0.9752\n",
      "Epoch 811/1000\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0241 - accuracy: 1.0000\n",
      "Epoch 812/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0616 - accuracy: 0.9752\n",
      "Epoch 813/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.2795 - accuracy: 0.9339\n",
      "Epoch 814/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.1635 - accuracy: 0.9587\n",
      "Epoch 815/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.0711 - accuracy: 0.9669\n",
      "Epoch 816/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1701 - accuracy: 0.9587\n",
      "Epoch 817/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0869 - accuracy: 0.9669\n",
      "Epoch 818/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0675 - accuracy: 0.9669\n",
      "Epoch 819/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1615 - accuracy: 0.9587\n",
      "Epoch 820/1000\n",
      "1/1 [==============================] - 0s 6ms/step - loss: 0.1472 - accuracy: 0.9669\n",
      "Epoch 821/1000\n",
      "1/1 [==============================] - 0s 18ms/step - loss: 0.0906 - accuracy: 0.9587\n",
      "Epoch 822/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1105 - accuracy: 0.9669\n",
      "Epoch 823/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.3633 - accuracy: 0.9256\n",
      "Epoch 824/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.2328 - accuracy: 0.9339\n",
      "Epoch 825/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1512 - accuracy: 0.9587\n",
      "Epoch 826/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1546 - accuracy: 0.9421\n",
      "Epoch 827/1000\n",
      "1/1 [==============================] - 0s 20ms/step - loss: 0.1470 - accuracy: 0.9669\n",
      "Epoch 828/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0739 - accuracy: 0.9587\n",
      "Epoch 829/1000\n",
      "1/1 [==============================] - 0s 428us/step - loss: 0.1807 - accuracy: 0.9587\n",
      "Epoch 830/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1613 - accuracy: 0.9421\n",
      "Epoch 831/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.2190 - accuracy: 0.9669\n",
      "Epoch 832/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0711 - accuracy: 0.9669\n",
      "Epoch 833/1000\n",
      "1/1 [==============================] - 0s 10ms/step - loss: 0.1439 - accuracy: 0.9339\n",
      "Epoch 834/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.1449 - accuracy: 0.9669\n",
      "Epoch 835/1000\n",
      "1/1 [==============================] - 0s 16ms/step - loss: 0.0923 - accuracy: 0.9669\n",
      "Epoch 836/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1667 - accuracy: 0.9256\n",
      "Epoch 837/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1297 - accuracy: 0.9669\n",
      "Epoch 838/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0851 - accuracy: 0.9835\n",
      "Epoch 839/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.1287 - accuracy: 0.9587\n",
      "Epoch 840/1000\n",
      "1/1 [==============================] - 0s 14ms/step - loss: 0.1728 - accuracy: 0.9504\n",
      "Epoch 841/1000\n",
      "1/1 [==============================] - 0s 12ms/step - loss: 0.1202 - accuracy: 0.9752\n",
      "Epoch 842/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.2377 - accuracy: 0.9669\n",
      "Epoch 843/1000\n",
      "1/1 [==============================] - 0s 13ms/step - loss: 0.1920 - accuracy: 0.9421\n",
      "Epoch 844/1000\n",
      "1/1 [==============================] - 0s 17ms/step - loss: 0.1875 - accuracy: 0.9504\n",
      "Epoch 845/1000\n",
      "1/1 [==============================] - 0s 19ms/step - loss: 0.2033 - accuracy: 0.9421\n",
      "Epoch 846/1000\n",
      "1/1 [==============================] - 0s 7ms/step - loss: 0.0765 - accuracy: 0.9752\n",
      "Epoch 847/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.0443 - accuracy: 1.0000\n",
      "Epoch 848/1000\n",
      "1/1 [==============================] - 0s 9ms/step - loss: 0.1209 - accuracy: 0.9587\n",
```

```
       "Epoch 849/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.1085 - accuracy: 0.9752\n",
       "Epoch 850/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.1361 - accuracy: 0.9587\n",
       "Epoch 851/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.0668 - accuracy: 0.9835\n",
       "Epoch 852/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.3633 - accuracy: 0.9174\n",
       "Epoch 853/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.2534 - accuracy: 0.9421\n",
       "Epoch 854/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1501 - accuracy: 0.9587\n",
       "Epoch 855/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.1013 - accuracy: 0.9587\n",
       "Epoch 856/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1388 - accuracy: 0.9587\n",
       "Epoch 857/1000\n",
       "1/1 [==============================] - 0s 6ms/step - loss: 0.1194 - accuracy: 0.9587\n",
       "Epoch 858/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1867 - accuracy: 0.9256\n",
       "Epoch 859/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.1070 - accuracy: 0.9587\n",
       "Epoch 860/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.1060 - accuracy: 0.9587\n",
       "Epoch 861/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.0916 - accuracy: 0.9752\n",
       "Epoch 862/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.0558 - accuracy: 0.9752\n",
       "Epoch 863/1000\n",
       "1/1 [==============================] - 0s 20ms/step - loss: 0.0859 - accuracy: 0.9752\n",
       "Epoch 864/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.1038 - accuracy: 0.9587\n",
       "Epoch 865/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.0763 - accuracy: 0.9752\n",
       "Epoch 866/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.1983 - accuracy: 0.9421\n",
       "Epoch 867/1000\n",
       "1/1 [==============================] - 0s 0s/step - loss: 0.1803 - accuracy: 0.9421\n",
       "Epoch 868/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.0438 - accuracy: 0.9917\n",
       "Epoch 869/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1067 - accuracy: 0.9587\n",
       "Epoch 870/1000\n",
       "1/1 [==============================] - 0s 13ms/step - loss: 0.1170 - accuracy: 0.9669\n",
       "Epoch 871/1000\n",
       "1/1 [==============================] - 0s 0s/step - loss: 0.2452 - accuracy: 0.9339\n",
       "Epoch 872/1000\n",
       "1/1 [==============================] - 0s 14ms/step - loss: 0.1926 - accuracy: 0.9421\n",
       "Epoch 873/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.2746 - accuracy: 0.9504\n",
       "Epoch 874/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.2377 - accuracy: 0.9504\n",
       "Epoch 875/1000\n",
       "1/1 [==============================] - 0s 14ms/step - loss: 0.1056 - accuracy: 0.9669\n",
       "Epoch 876/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.1332 - accuracy: 0.9752\n",
       "Epoch 877/1000\n",
       "1/1 [==============================] - 0s 14ms/step - loss: 0.0963 - accuracy: 0.9587\n",
       "Epoch 878/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.1037 - accuracy: 0.9587\n",
       "Epoch 879/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.0859 - accuracy: 0.9669\n",
       "Epoch 880/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.1415 - accuracy: 0.9504\n",
       "Epoch 881/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.1770 - accuracy: 0.9504\n",
       "Epoch 882/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.1326 - accuracy: 0.9421\n",
       "Epoch 883/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.0888 - accuracy: 0.9587\n",
       "Epoch 884/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.1288 - accuracy: 0.9587\n",
       "Epoch 885/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.1002 - accuracy: 0.9669\n",
       "Epoch 886/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.1256 - accuracy: 0.9587\n",
       "Epoch 887/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.0876 - accuracy: 0.9587\n",
       "Epoch 888/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.1514 - accuracy: 0.9669\n",
       "Epoch 889/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1818 - accuracy: 0.9504\n",
       "Epoch 890/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1820 - accuracy: 0.9339\n",
       "Epoch 891/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.0318 - accuracy: 0.9835\n",
       "Epoch 892/1000\n"
      ]
     },
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "1/1 [==============================] - 0s 10ms/step - loss: 0.1249 - accuracy: 0.9587\n",
       "Epoch 893/1000\n",
       "1/1 [==============================] - 0s 11ms/step - loss: 0.2429 - accuracy: 0.9504\n",
       "Epoch 894/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.0519 - accuracy: 0.9917\n",
       "Epoch 895/1000\n",
       "1/1 [==============================] - 0s 13ms/step - loss: 0.0764 - accuracy: 0.9752\n",
       "Epoch 896/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.1840 - accuracy: 0.9421\n",
       "Epoch 897/1000\n",
       "1/1 [==============================] - 0s 11ms/step - loss: 0.0477 - accuracy: 0.9835\n",
```

```
"Epoch 898/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0596 - accuracy: 0.9835\n",
"Epoch 899/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0705 - accuracy: 0.9669\n",
"Epoch 900/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.2075 - accuracy: 0.9421\n",
"Epoch 901/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1416 - accuracy: 0.9587\n",
"Epoch 902/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.1808 - accuracy: 0.9256\n",
"Epoch 903/1000\n",
"1/1 [==============================] - 0s 21ms/step - loss: 0.0326 - accuracy: 0.9835\n",
"Epoch 904/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0988 - accuracy: 0.9587\n",
"Epoch 905/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0509 - accuracy: 0.9752\n",
"Epoch 906/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.0769 - accuracy: 0.9587\n",
"Epoch 907/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1246 - accuracy: 0.9504\n",
"Epoch 908/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1575 - accuracy: 0.9587\n",
"Epoch 909/1000\n",
"1/1 [==============================] - 0s 16ms/step - loss: 0.1931 - accuracy: 0.9669\n",
"Epoch 910/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1217 - accuracy: 0.9504\n",
"Epoch 911/1000\n",
"1/1 [==============================] - 0s 6ms/step - loss: 0.0795 - accuracy: 0.9669\n",
"Epoch 912/1000\n",
"1/1 [==============================] - 0s 20ms/step - loss: 0.0824 - accuracy: 0.9669\n",
"Epoch 913/1000\n",
"1/1 [==============================] - 0s 9ms/step - loss: 0.0837 - accuracy: 0.9587\n",
"Epoch 914/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.0627 - accuracy: 0.9752\n",
"Epoch 915/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0966 - accuracy: 0.9587\n",
"Epoch 916/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.2907 - accuracy: 0.9339\n",
"Epoch 917/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1094 - accuracy: 0.9835\n",
"Epoch 918/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.1043 - accuracy: 0.9669\n",
"Epoch 919/1000\n",
"1/1 [==============================] - 0s 10ms/step - loss: 0.1320 - accuracy: 0.9835\n",
"Epoch 920/1000\n",
"1/1 [==============================] - 0s 12ms/step - loss: 0.1127 - accuracy: 0.9752\n",
"Epoch 921/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.0706 - accuracy: 0.9669\n",
"Epoch 922/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.0694 - accuracy: 0.9669\n",
"Epoch 923/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.1558 - accuracy: 0.9752\n",
"Epoch 924/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0816 - accuracy: 0.9752\n",
"Epoch 925/1000\n",
"1/1 [==============================] - 0s 14ms/step - loss: 0.1761 - accuracy: 0.9504\n",
"Epoch 926/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.2439 - accuracy: 0.9669\n",
"Epoch 927/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0795 - accuracy: 0.9752\n",
"Epoch 928/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1763 - accuracy: 0.9504\n",
"Epoch 929/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0990 - accuracy: 0.9669\n",
"Epoch 930/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.2355 - accuracy: 0.9421\n",
"Epoch 931/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0937 - accuracy: 0.9669\n",
"Epoch 932/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.1018 - accuracy: 0.9504\n",
"Epoch 933/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.2176 - accuracy: 0.9587\n",
"Epoch 934/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.1502 - accuracy: 0.9587\n",
"Epoch 935/1000\n",
"1/1 [==============================] - 0s 12ms/step - loss: 0.1219 - accuracy: 0.9504\n",
"Epoch 936/1000\n",
"1/1 [==============================] - 0s 880us/step - loss: 0.1206 - accuracy: 0.9504\n",
"Epoch 937/1000\n",
"1/1 [==============================] - 0s 8ms/step - loss: 0.0975 - accuracy: 0.9504\n",
"Epoch 938/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.0618 - accuracy: 0.9752\n",
"Epoch 939/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.1658 - accuracy: 0.9504\n",
"Epoch 940/1000\n",
"1/1 [==============================] - 0s 7ms/step - loss: 0.2141 - accuracy: 0.9504\n",
"Epoch 941/1000\n",
"1/1 [==============================] - 0s 19ms/step - loss: 0.2479 - accuracy: 0.9504\n",
"Epoch 942/1000\n",
"1/1 [==============================] - 0s 11ms/step - loss: 0.1068 - accuracy: 0.9669\n",
"Epoch 943/1000\n",
"1/1 [==============================] - 0s 6ms/step - loss: 0.0911 - accuracy: 0.9752\n",
"Epoch 944/1000\n",
"1/1 [==============================] - 0s 13ms/step - loss: 0.0441 - accuracy: 0.9835\n",
"Epoch 945/1000\n",
"1/1 [==============================] - 0s 15ms/step - loss: 0.0441 - accuracy: 0.9835\n",
"Epoch 946/1000\n",
"1/1 [==============================] - 0s 515us/step - loss: 0.1382 - accuracy: 0.9669\n",
"Epoch 947/1000\n",
"1/1 [==============================] - 0s 17ms/step - loss: 0.2314 - accuracy: 0.9504\n",
"Epoch 948/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0332 - accuracy: 1.0000\n",
"Epoch 949/1000\n",
"1/1 [==============================] - 0s 0s/step - loss: 0.0438 - accuracy: 0.9835\n",
```

```
       "Epoch 950/1000\n",
       "1/1 [==============================] - 0s 14ms/step - loss: 0.1111 - accuracy: 0.9587\n",
       "Epoch 951/1000\n",
       "1/1 [==============================] - 0s 11ms/step - loss: 0.1608 - accuracy: 0.9504\n",
       "Epoch 952/1000\n",
       "1/1 [==============================] - 0s 0s/step - loss: 0.1148 - accuracy: 0.9752\n",
       "Epoch 953/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.0894 - accuracy: 0.9669\n",
       "Epoch 954/1000\n",
       "1/1 [==============================] - 0s 19ms/step - loss: 0.1470 - accuracy: 0.9587\n",
       "Epoch 955/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.2286 - accuracy: 0.9256\n",
       "Epoch 956/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.1207 - accuracy: 0.9752\n",
       "Epoch 957/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1240 - accuracy: 0.9752\n",
       "Epoch 958/1000\n",
       "1/1 [==============================] - 0s 12ms/step - loss: 0.0608 - accuracy: 0.9835\n",
       "Epoch 959/1000\n",
       "1/1 [==============================] - 0s 13ms/step - loss: 0.1882 - accuracy: 0.9339\n",
       "Epoch 960/1000\n",
       "1/1 [==============================] - 0s 12ms/step - loss: 0.1118 - accuracy: 0.9669\n",
       "Epoch 961/1000\n",
       "1/1 [==============================] - 0s 19ms/step - loss: 0.2058 - accuracy: 0.9339\n",
       "Epoch 962/1000\n",
       "1/1 [==============================] - 0s 0s/step - loss: 0.1674 - accuracy: 0.9504\n",
       "Epoch 963/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.1171 - accuracy: 0.9587\n",
       "Epoch 964/1000\n",
       "1/1 [==============================] - 0s 0s/step - loss: 0.1669 - accuracy: 0.9421\n",
       "Epoch 965/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.0878 - accuracy: 0.9669\n",
       "Epoch 966/1000\n",
       "1/1 [==============================] - 0s 12ms/step - loss: 0.1307 - accuracy: 0.9256\n",
       "Epoch 967/1000\n",
       "1/1 [==============================] - 0s 1ms/step - loss: 0.0752 - accuracy: 0.9587\n",
       "Epoch 968/1000\n",
       "1/1 [==============================] - 0s 14ms/step - loss: 0.1108 - accuracy: 0.9587\n",
       "Epoch 969/1000\n",
       "1/1 [==============================] - 0s 11ms/step - loss: 0.0708 - accuracy: 0.9587\n",
       "Epoch 970/1000\n",
       "1/1 [==============================] - 0s 12ms/step - loss: 0.1195 - accuracy: 0.9587\n",
       "Epoch 971/1000\n",
       "1/1 [==============================] - 0s 10ms/step - loss: 0.0534 - accuracy: 0.9835\n",
       "Epoch 972/1000\n",
       "1/1 [==============================] - 0s 1ms/step - loss: 0.1602 - accuracy: 0.9669\n",
       "Epoch 973/1000\n"
      ]
     },
     {
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "1/1 [==============================] - 0s 11ms/step - loss: 0.1811 - accuracy: 0.9339\n",
       "Epoch 974/1000\n",
       "1/1 [==============================] - 0s 744us/step - loss: 0.1300 - accuracy: 0.9421\n",
       "Epoch 975/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.1105 - accuracy: 0.9752\n",
       "Epoch 976/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.1176 - accuracy: 0.9504\n",
       "Epoch 977/1000\n",
       "1/1 [==============================] - 0s 18ms/step - loss: 0.1182 - accuracy: 0.9669\n",
       "Epoch 978/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.1389 - accuracy: 0.9835\n",
       "Epoch 979/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.0855 - accuracy: 0.9504\n",
       "Epoch 980/1000\n",
       "1/1 [==============================] - 0s 18ms/step - loss: 0.0945 - accuracy: 0.9669\n",
       "Epoch 981/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1463 - accuracy: 0.9587\n",
       "Epoch 982/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.1928 - accuracy: 0.9339\n",
       "Epoch 983/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1465 - accuracy: 0.9421\n",
       "Epoch 984/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1640 - accuracy: 0.9421\n",
       "Epoch 985/1000\n",
       "1/1 [==============================] - 0s 7ms/step - loss: 0.0831 - accuracy: 0.9752\n",
       "Epoch 986/1000\n",
       "1/1 [==============================] - 0s 0s/step - loss: 0.1167 - accuracy: 0.9421\n",
       "Epoch 987/1000\n",
       "1/1 [==============================] - 0s 16ms/step - loss: 0.1849 - accuracy: 0.9421\n",
       "Epoch 988/1000\n",
       "1/1 [==============================] - 0s 17ms/step - loss: 0.0456 - accuracy: 0.9835\n",
       "Epoch 989/1000\n",
       "1/1 [==============================] - 0s 10ms/step - loss: 0.2812 - accuracy: 0.9339\n",
       "Epoch 990/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.0916 - accuracy: 0.9587\n",
       "Epoch 991/1000\n",
       "1/1 [==============================] - 0s 20ms/step - loss: 0.0929 - accuracy: 0.9917\n",
       "Epoch 992/1000\n",
       "1/1 [==============================] - 0s 8ms/step - loss: 0.1539 - accuracy: 0.9504\n",
       "Epoch 993/1000\n",
       "1/1 [==============================] - 0s 20ms/step - loss: 0.1329 - accuracy: 0.9421\n",
       "Epoch 994/1000\n",
       "1/1 [==============================] - 0s 9ms/step - loss: 0.1356 - accuracy: 0.9669\n",
       "Epoch 995/1000\n",
       "1/1 [==============================] - 0s 13ms/step - loss: 0.1472 - accuracy: 0.9339\n",
       "Epoch 996/1000\n",
       "1/1 [==============================] - 0s 12ms/step - loss: 0.0594 - accuracy: 0.9752\n",
       "Epoch 997/1000\n",
       "1/1 [==============================] - 0s 15ms/step - loss: 0.1118 - accuracy: 0.9669\n",
       "Epoch 998/1000\n",
       "1/1 [==============================] - 0s 18ms/step - loss: 0.2219 - accuracy: 0.9421\n",
```

```
      "Epoch 999/1000\n",
      "1/1 [==============================] - 0s 8ms/step - loss: 0.0738 - accuracy: 0.9587\n",
      "Epoch 1000/1000\n",
      "1/1 [==============================] - 0s 15ms/step - loss: 0.1649 - accuracy: 0.9587\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "<keras.callbacks.History at 0x242e78768e0>"
      ]
     },
     "execution_count": 25,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "#building neural network model,we'll create Keras? Sequential model.\n",
    "#The input to this network will be the array train_X\n",
    "# the model has 3 layers with the first having 128 neurons, the second having 64 neurons, and the last layer having th
    "# Next, to reach the correct weights, Adam optimizer is used. loss function is  categorical cross-entropy function.\n"
    "# And, the metric  to evaluate the model is 'accuracy'.\n",
    "import tensorflow as tf\n",
    "from tensorflow.keras import Sequential\n",
    "from tensorflow.keras.layers import Dense, Dropout\n",
    "model = Sequential()\n",
    "# the number of units(neurons) could affect the resulted training accuracy\n",
    "model.add(Dense(128, input_shape =(len(train_x[0]),), activation ='relu'))#input layer\n",
    "model.add(Dropout(0.5))#dropout layer\n",
    "model.add(Dense(64, activation ='relu'))#hidden layer\n",
    "model.add(Dropout(0.5))\n",
    "# output layer, classification model(activation function: softmax)\n",
    "model.add(Dense(len(train_y[0]), activation ='softmax'))\n",
    "adam =tf.keras.optimizers.Adam(learning_rate=0.01)    \n",
    "model.compile(loss= 'categorical_crossentropy', \n",
    "              optimizer =adam,\n",
    "              metrics=['accuracy'])\n",
    "print(model.summary)\n",
    "model.fit(x=train_x, y=train_y, epochs=1000, batch_size=128)\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "362ed2a1",
   "metadata": {},
   "outputs": [],
   "source": [
    "# model evaluation with unknown data\n",
    "# chat preparation functions\n",
    "def sent_clean(sent):\n",
    "    # tokenizing the pattern\n",
    "    tokens = nltk.word_tokenize(sent)\n",
    "    # stemming each word\n",
    "    sent_words = [stemmer.stem(word.lower()) for word in tokens]\n",
    "    return sent_words\n",
    "\n",
    "# returning bag of words of the pattern\n",
    "def bw(sent, words):\n",
    "    # tokenizing the pattern\n",
    "    sent_words = sent_clean(sent)\n",
    "    # create bag of words\n",
    "    bag = [0]*len(words)  \n",
    "    for s in sent_words:\n",
    "        for i,w in enumerate(words):\n",
    "            if w == s: \n",
    "                bag[i] = 1\n",
    "                \n",
    "\n",
    "    return(np.array(bag))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "0cdbe099",
   "metadata": {},
   "outputs": [],
   "source": [
    "# classify the pattern with a tag, and generate the appropriate response\n",
    "error_threshold = 0.30\n",
    "def classify(sent):\n",
    "    # predict class probabilities from the model\n",
    "    #tag_prob = model.predict(np.array([bw(sent, words)]))[0]# for sequential model \n",
    "    tag_prob = model.predict([bw(sent, words)])[0] # for dnn model\n",
    "    # filter out predictions below a threshold\n",
    "    results = [[i,r] for i,r in enumerate(tag_prob) if r > error_threshold]\n",
    "    # sort probabilities in descending order\n",
    "    results.sort(key=lambda x: x[1], reverse=True)\n",
    "    return_list = []\n",
    "    for r in results:\n",
    "        return_list.append((classes[r[0]], r[1]))# get the tag of index r[0] und prob_value is r[1]\n",
    "    # return tuple of intent and probability\n",
    "    return return_list\n",
    "\n",
    "def response(sentence):\n",
    "    results = classify(sentence)\n",
    "    # if we have a classification then find the matching intent tag\n",
    "    if results:\n",
    "        # loop as long as there are matches to process\n",
    "        while results:\n",
    "            for chat in chats['intents']:\n",
    "                # find a tag matching the first result (highst probability)\n",
```

```
    "                if chat['tag'] == results[0][0]:\n",
    "                    # print out a random response of the corresponding tag\n",
    "                    return print(random.choice(chat['responses']))\n",
    "\n",
    "            results.pop(0)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "20f0f465",
   "metadata": {},
   "outputs": [],
   "source": [
    "print('enter 0 if you dont want to chat with this Chatbot')\n",
    "while True:\n",
    "    \n",
    "    message = input('')\n",
    "    if message==0:\n",
    "        break\n",
    "    res = response(message)\n",
    "    print(res)\n",
    " "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "0353b800",
   "metadata": {},
   "outputs": [],
   "source": [
    "\n"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.13"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

intents.json (11.79 KB)

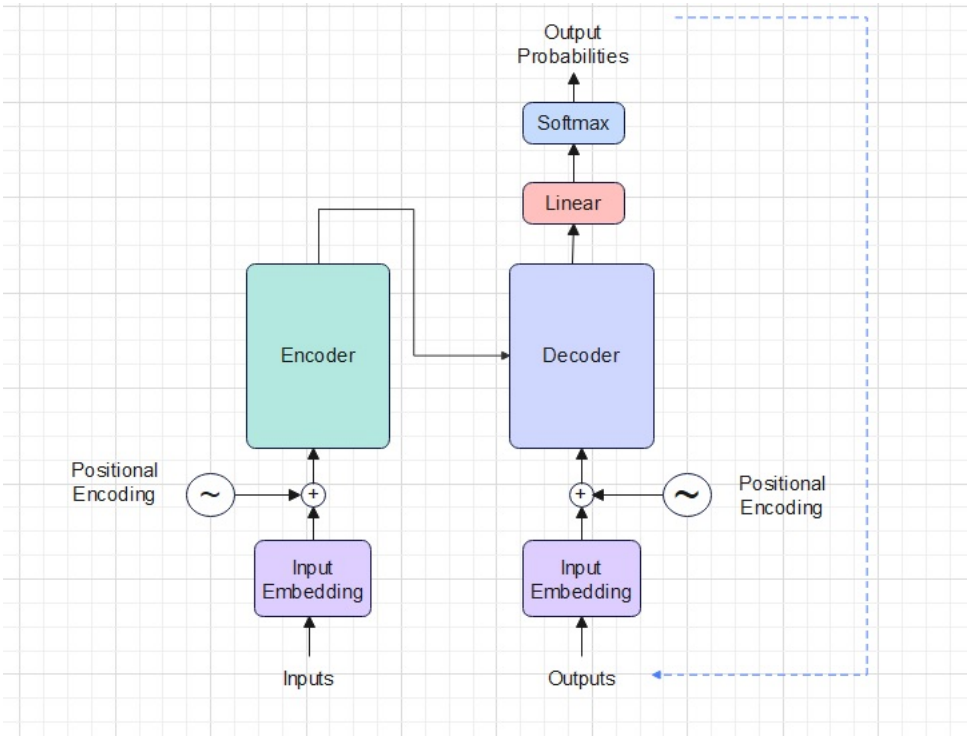
Transformers

Der Transformer im NLP ist eine neue Technologie, die darauf abzielt, Sequenz-zu-Sequenz-Probleme durch die Einbeziehung langfristiger Abhängigkeiten zu lösen. Es verwendet das Selbstaufmerksamkeitskonzept, um den Kontext der Eingabesequenz zu kodieren und einzubetten.

Beim Deep Learning ist Aufmerksamkeit ein neues Konzept, das es ermöglicht, auf Teile der Eingabesequenz zu konzentrieren, Um ein Verständnis des Kontexts als Ergebnis vermittelt zu werden. Und dies ist eine Art von Aufmerksamkeit in einem Transformatormodell namens Encoder-Decoder-Aufmerksamkeit, bei dem eine Verbindung/Korrelation zwischen Eingabe und Ausgabe abgebildet wird. Bei einer anderen Art der Aufmerksamkeit namens Selbstaufmerksamkeit erstellt das Modell ähnliche Verbindungen, jedoch innerhalb eines einzelnen Satzes, um eine Darstellung der Sequenz zu berechnen. Der dritte Typ wird als Selbstaufmerksamkeit in der Ausgabesequenz bezeichnet, wobei die Aufmerksamkeit hier nur auf die Wörter vor einem bestimmten Wort beschränkt ist, während die übrigen Wörter maskiert sind.

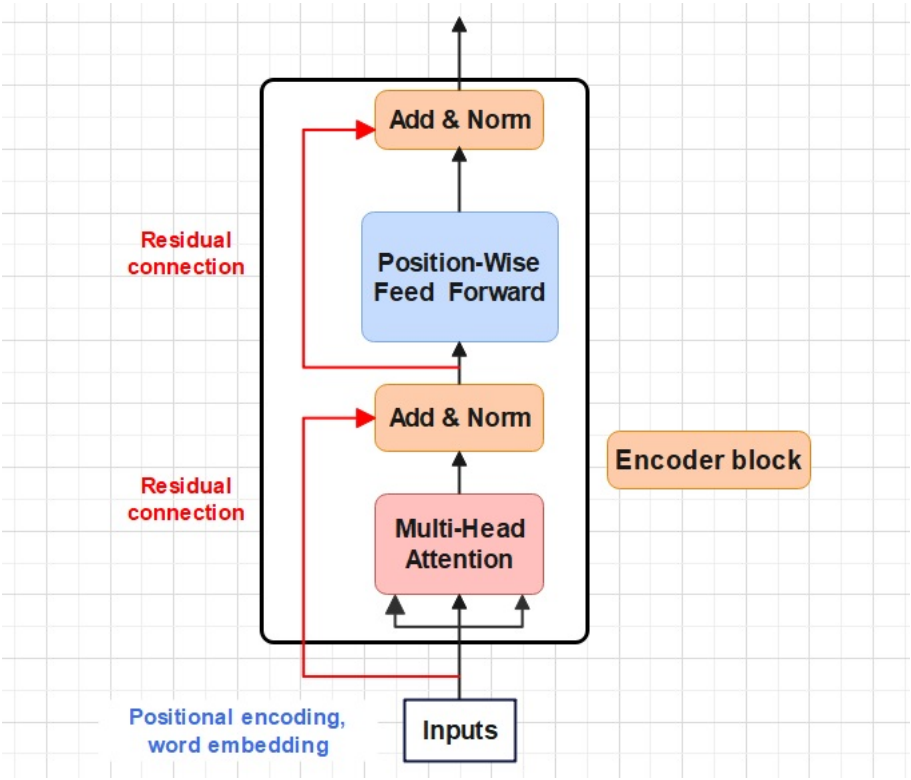
Encoder-Decoder-Transformator-Architektur:

Die folgende Abbildung zeigt einen Encoder-Decoder-Transformator. Die Aufgabe des Encoders auf der linken Seite besteht darin, eine Eingabesequenz Darstellungen zuzuordnen, die vom Decoder zusammen mit der Decoderausgabe im vorherigen Zeitschritt verwendet werden, um eine Ausgabesequenz zu generieren.



Einbettungen sind die Kernidee des Transformators für neuronale maschinelle Übersetzungsaufgaben (NMT). Durch Einbettungen kann das Modell mit mehr Informationen über die Wörter versorgt werden und das Modell kann effizienter über den Kontext lernen. Dies funktioniert, indem nach dem Einbettungsvektor für jedes in das Netzwerk eingespeiste Wort gesucht wird. Die Bedeutung der Wörter wird so kodiert, dass Wörter mit ähnlicher Bedeutung in ähnlichen Vektoren dargestellt werden.

Die Positionskodierung informiert das Netzwerk über die Positionen der Wörter oder darüber, wo sich das Wort im Satz befindet. Der Encoder verwendet diese Codierung zusammen mit den Einbettungen der Wörter, um über mehrere Encoderblöcke lineare Algebraoperationen an den Einbettungsvektoren durchzuführen, um Merkmale über die Kontexte für jedes Wort innerhalb des gesamten Satzes zu extrahieren und die Einbettungsvektoren mit diesen Merkmalen anzureichern.

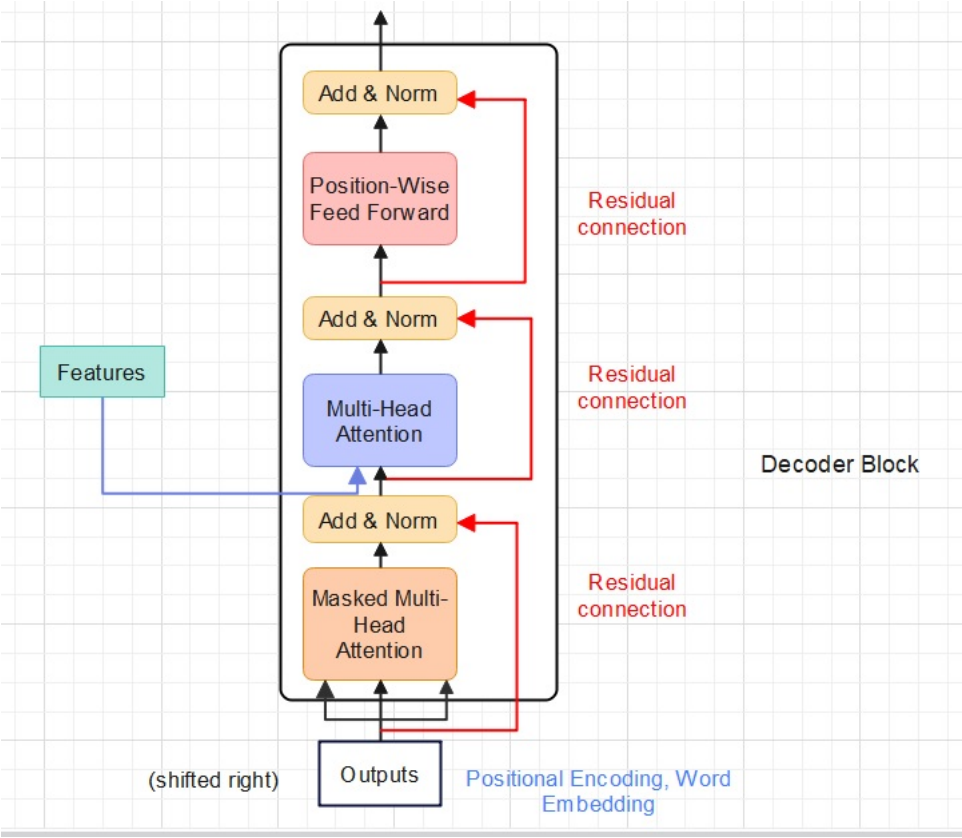


Der Encoderblock nutzt den Selbstaufmerksamkeitsmechanismus, um jeden Einbettungsvektor mit Kontextinformationen des Tokens in seinem Satz anzureichern. Da ein Wortvektor mehrere Semantiken enthalten kann, werden mehrere Köpfe verwendet, sodass jeder Kopf den Vektor mit seiner eigenen Matrix projiziert, um eine sematische Eigenschaft zu überprüfen. Beispielsweise kann das Wort „Königin" mehrere semantische Eigenschaften haben, etwa weiblich, königlich oder Macht.

Das positionsbezogene Feed-Forward-Netzwerk verfügt über zwei lineare Schichten, die zusammen mit Kontextinformationen aus der Multi-Head-Aufmerksamkeit eine Transformation auf jeden Einbettungsvektor anwenden. Die Hauptaufgabe dieses Netzwerks besteht darin, jeden Worteinbettungsvektor unabhängig mit identischen Gewichten basierend auf seiner Position in der Sequenz zu verarbeiten.

Restverbindungen sorgen dafür, dass die Token-Einbettungen und Positionsinformationen von der vorherigen Schicht zur weiteren Verarbeitung in die nachfolgende Schicht übertragen werden.

Die Schichtnormalisierung wird verwendet, um das Training zu stabilisieren, indem verhindert wird, dass sich der Mittelwert und die Standardabweichung des Einbettungsvektors verschieben. Das heißt, alle Elemente im Einbettungsvektor auf den gleichen Maßstab zu bringen, wodurch das Netzwerk schneller konvergieren und trainieren kann.



Der Decoder verwendet die vom Encoder generierten Mermale, um eine Ausgabe zu erzeugen. Die Eingabe in den Decoder ist die nach rechts verschobene Ausgabe des vorherigen Zeitschritts, die zu einer Folge von positionscodierten Einbettungen wird. Die maskierte Multi-Head-Aufmerksamkeit ist eine wichtige Ebene im Decoder, die die Eingabe mit Masken versorgt, um der nächsten Multi-Head-Aufmerksamkeitsschicht dabei zu helfen, die maskierten Positionen in den vom Encoder berechneten Merkmalen zu verbergen. Der Decoder erreicht diesen Schritt, indem er die Aufmerksamkeitswerte der maskierten Positionen auf negative Werte (-∞) setzt.

Die Multi-Head-Aufmerksamkeitsschicht, die auch als Quelle-Ziel-Aufmerksamkeit bekannt ist, berechnet anschließend Aufmerksamkeitsmerkmale zwischen Eingabeeinbettungen und der Ausgabe aus dem vorherigen Zeitschritt. Die aus dieser Schicht resultierenden Merkmalsvektoren sind für das Training erforderlich, da sie der nächsten Feed-Forward-Schicht dabei helfen, neue Gewichte des Netzwerks zu lernen.

Anschließend wendet der Decoder eine lineare Transformation auf den Ausgabevektor an, um die Größe eines Einbettungsvektors in einen Vokabulargrößenvektor zu ändern. Softmax wird hier verwendet, um den resultierenden Logit-Vector in einen Wahrscheinlichkeitsvektor mit der gleichen Größe wie das Korpusvokabular zu skalieren.

Die Ausgabe des Decoders ist jeweils ein Token. Das Ausgabetoken wird dann als nachfolgende Eingabe für den Decoder verwendet, nachdem es durch die Einbettungsschicht in einen Einbettungsvektor umgewandelt wurde.

Transformer kann für viele NLP-Aufgaben verwendet werden, einschließlich Übersetzung und Textklassifizierung.

Transformers Übung

Bevor wir mit der Verwendung von Transformatoren beginnen, müssen wir alle erforderlichen Bibliotheken installieren:

```
In [2]: # installing transformer library
        !pip install transformers datasets evaluate
```

```
Requirement already satisfied: transformers in c:\users\la2022\anaconda3\lib\site-packages (4.25.1)
Collecting datasets
  Downloading datasets-2.9.0-py3-none-any.whl (462 kB)
     ---------------------------------------- 462.8/462.8 kB 5.8 MB/s eta 0:00:00
Collecting evaluate
  Downloading evaluate-0.4.0-py3-none-any.whl (81 kB)
     ---------------------------------------- 81.4/81.4 kB 4.4 MB/s eta 0:00:00
Requirement already satisfied: filelock in c:\users\la2022\anaconda3\lib\site-packages (from transformers) (3.6.0)
Requirement already satisfied: packaging>=20.0 in c:\users\la2022\anaconda3\lib\site-packages (from transformers) (21.3)
```

In diesem Beispiel wenden wir einen Transformator für die Textklassifizierungsaufgabe im E-Commerce-Datensatz hier an. Die Datensätze bestehen aus zwei Spalten: Produktbeschreibung, die einige Merkmale und Attribute des Produkts enthält, während die zweite Spalte die Produktklasse mit vier Hauptkategorien enthält: „Elektronik", „Haushalt", „Bücher" und „Kleidung und Accessoires". Die Kategorien der Produkte müssen in Zahlenwerte umgerechnet werden.

In [22]:
```
# installing dataset
import pandas as pd
from sklearn.utils import shuffle

data = pd.read_csv("ecomm_data.csv")
# remaove na values
data = data.dropna()
# shuffle data
data = shuffle(data)
# rename column names using col index
data.rename(columns={data.columns[0]: "label", data.columns[1]: "product_description"}, inplace=True)
print(data.head(1)['product_description'])

# replace catecorical with numerical values
data['label'].replace([ "Electronics",  "Household", "Books", "Clothing & Accessories"],
                      [0, 1, 2, 3], inplace=True)
```

```
5882    Elite Home 600SSQU053DLHS6 Sheet Set, Queen Si...
Name: product_description, dtype: object
```

Der nächste Schritt umfasst die Konvertierung des Wörterbuchs in einen Datensatz, damit dieser anschließend vom Tokenizer verwendet werden kann. Nur ein Teil des Datensatzes wird für das Training des Transformators verwendet. Um die Wörter der Trainingsdaten zu tokenisieren, verwenden wir einen vorab trainierten Ditilbert-Tokenizer, um Token zu erzeugen, die dem Typ des Transformators entsprechen, in unserem Fall des Ditilbert-Transformators. Zur Tokenisierung müssen wir Eingabesequenzen auf die maximale Eingabelänge von DistilBERT kürzen.

In [24]:
```
# convert data into dataset
df = {'product_desc': data['product_description'], 'label': data['label']}
from datasets import Dataset
out_df = Dataset.from_dict(df)

print(out_df[:2])

#load a DistilBERT tokenizer, tokeniztation of the words
from transformers import AutoTokenizer
# using the distilbert tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased",return_tensors='pt')

#tokenization function
# truncate sequences longer than DistilBERT's maximum input length

def tokenize(data):
    return tokenizer(data["product_desc"], truncation=True)

#splitting dataset into train and test:
train = out_df[:1000]
test =  out_df[1000:1100]
train = Dataset.from_dict(train)
test = Dataset.from_dict(test)


print(train[:2])
print(test[:2])
```

In [6]:
```
# tokenizing train and test sets using map function, to avoid creating tokenizers.Encoding objects and keep dict or Dataset
# objects with keys
train_tokens = train.map(tokenize, batched=True)
test_tokens= test.map(tokenize, batched=True)
```

```
100% 1/1 [00:00<00:00, 3.64ba/s]

100% 1/1 [00:00<00:00, 21.59ba/s]
```

Wie beim Kürzungsschritt müssen wir Padding verwenden, um Eingabestapel mit der gleichen Länge zu erzeugen. Dies wird mit DataCollatorWithPadding durchgeführt, das die empfangenen Eingaben dynamisch auffüllt.

In [7]:
```
# datacollatorwithpadding: to create batches with padding to the maximum input length; pytorch implementation..
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Als nächstes definieren wir eine Genauigkeitsfunktion, um die Vorhersageleistung des Transformators zu bewerten. Zu diesem Zweck verwenden wir eine „Genauigkeits"-Metrik, die als Verhältnis der korrekten Vorhersagen zur Gesamtzahl der Vorhersagen berechnet wird. Das Zuordnen vorhergesagter IDs zu ihren Beschriftungen ist ebenfalls ein notwendiger Schritt, der es dem Transformator ermöglicht, Vorhersagen oder Klassen als Texte statt als Zahlen auszugeben.

In [25]:
```
# defining a function to calculate prediction performance of the model:  'accuracy' metric will be used

import numpy as np
import evaluate

accuracy = evaluate.load("accuracy")
def acc_calculate(pred):
    predicted, actual = pred
    predicted = np.argmax(predicted, axis=1)# argmax returns indices of the maximum values along axis 1;
    # predicted contains the probabilities of all classes for each entry in the dataset,
    # the index 0 means that probabilty of class 0 is the highst and so on..
    return accuracy.compute(predictions=predicted, references=actual)

# create a map of the expected ids to their labels with id2label and label2id:

id2label = {0: "Electronics", 1: "Household", 2: "Books", 3:"Clothing & Accessories"}
label2id = {"Electronics": 0, "Household": 1, "Books":2, "Clothing & Accessories":3}
```

Jetzt ist es an der Zeit, unseren Transformator zu bauen und zu trainieren. Wir werden den vorab trainierten Distilbert-Transformator mithilfe des Python-Pakets AutoModelForSequenceClassification definieren, einem generischen Sequenzklassifizierungsmodell, das zum Initiieren des

Transformators mit einem vorab trainierten Modell verwendet wird. Für unser Beispiel verwenden wir die Gewichte des vorab trainierten Bert-Modells. Der Transformator wird an einem 3-Phasen-Trainingsset trainiert und anschließend mithilfe von Testtokens wie folgt bewertet:

In [10]:
```
# defining the transformer
from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer

#from_pretrained(): to load the pretrained Bert model weights
#AutoModelForSequenceClassification: is a generic sequence classification model that will be instantiated using a pretarined Bert

transformer = AutoModelForSequenceClassification.from_pretrained(
"distilbert-base-uncased", num_labels=4, id2label=id2label, label2id=label2id, from_tf=True
)
```

```
All TF 2.0 model weights were used when initializing DistilBertForSequenceClassification.

All the weights of DistilBertForSequenceClassification were initialized from the TF 2.0 model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use DistilBertForSequenceClassi
fication for predictions without further training.
```

In [ ]:
```
# defining the training parameters
training_pars = TrainingArguments(
    output_dir="my_transformer",
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
    num_train_epochs=3,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True)

# defining the trainer of the model
#The Trainer class provides an API for feature-complete training in PyTorch
trainer = Trainer(
    model=transformer,
    args=training_pars,
    train_dataset= train_tokens,
    eval_dataset=test_tokens,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=acc_calculate,
)
#training the transformer
trainer.train()
```

In [27]:
```
trainer.evaluate()
```

```
The following columns in the evaluation set don't have a corresponding argument in `DistilBertForSequenceClassification.forward
` and have been ignored: product_desc. If product_desc are not expected by `DistilBertForSequenceClassification.forward`,  you
can safely ignore this message.
***** Running Evaluation *****
  Num examples = 100
  Batch size = 64
```

Out[27]:
```
{'eval_loss': 0.3000350892543793, 'eval_accuracy': 0.91}
```

Für die zukünftige Verwendung können wir den trainierten Transformator in einer Datei speichern. Um Vorhersagen für neue Dateneingaben zu treffen, können wir die Transformer-Pipeline verwenden, die eine Wrapper-Klasse anderer Pipelines ist, in unserem Fall die Textklassifizierung. Diese Pipeline wird verwendet, um Eingabetext basierend auf dem trainierten Transfromer zu klassifizieren. Das Argument „return_all_scores" ist auf „true" gesetzt, sodass wir alle Bewertungen für jedes Label einer Vorhersage erhalten. Hier ist ein Beispiel:

In [ ]:
```
#save trainer to a file
trainer.save_model("./Distilbert_based_transformer")
transformer_trained = AutoModelForSequenceClassification.from_pretrained("./Distilbert_based_transformer")
```

In [26]:
```
from transformers import TextClassificationPipeline
# This pipeline has a return_all_scores parameter on its __call__ method that allows you to get all scores for each label on a pr
pipe = TextClassificationPipeline(model=transformer_trained, tokenizer=tokenizer)
prediction = pipe("NetGen 4k Wi-Fi 16 MP Ultra HD Action Camera", return_all_scores=True)
prediction
```

```
C:\Users\la2022\anaconda3\lib\site-packages\transformers\pipelines\text_classification.py:104: UserWarning: `return_all_scores`
is now deprecated,  if want a similar funcionality use `top_k=None` instead of `return_all_scores=True` or `top_k=1` instead of
`return_all_scores=False`.
  warnings.warn(
```

Out[26]:
```
[[{'label': 'Electronics', 'score': 0.9508970975875854},
  {'label': 'Household', 'score': 0.017261911183595657},
  {'label': 'Books', 'score': 0.017655229195952415},
  {'label': 'Clothing & Accessories', 'score': 0.014185710810124874}]]
```

Jetzt werden wir das vorab trainierte GPT 2-Modell verwenden, um Texte zu generieren:

In [ ]:
```
# using GPT-2 to generate text
from transformers import pipeline
text_generator = pipeline('text-generation', model='gpt2')
# generate 5 different sentences by sampling from the top 10 hits:
#Temperature is a hyper-parameter used to control the randomness of predictions by scaling the logits before applying softmax.
# when temperature is a large value (e.g. 1), the GPT-2 model produces more diversity and also more mistakes and viseversa
```

In [ ]:
```
text_generator.save_pretrained('gpt2_pretrained')
```

In [ ]:
```
sentences = text_generator("in the last few years, a bunch of changes", do_sample=True, top_k=10, temperature=0.6, max_length=50,
for sent in sentences:
  print(sent["generated_text"])
  print("#"*50)
```

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "a4315477",
   "metadata": {},
   "outputs": [],
   "source": [
    "# installing transformer library\n",
    "!pip install transformers datasets evaluate"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "b0a3c8dc",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "42997    TP-Link TL-UE300 USB 3.0 to RJ45 Gigabit Ether...\n",
      "Name: product_description, dtype: object\n"
     ]
    }
   ],
   "source": [
    "# installing dataset\n",
    "import pandas as pd\n",
    "from sklearn.utils import shuffle\n",
    "from transformers import AutoModelForSequenceClassification, TrainingArguments, Trainer\n",
    "data = pd.read_csv(\"ecomm_data.csv\")\n",
    "# remaove na values\n",
    "data = data.dropna()\n",
    "# shuffle data\n",
    "data = shuffle(data)\n",
    "# rename column names using col index\n",
    "data.rename(columns={data.columns[0]: \"label\", data.columns[1]: \"product_description\"}, inplace=True)\n",
    "print(data.head(1)['product_description'])\n",
    "\n",
    "# replace catecorical with numerical values\n",
    "data['label'].replace([ \"Electronics\",  \"Household\", \"Books\", \"Clothing & Accessories\"],\n",
    "                       [0, 1, 2, 3], inplace=True)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "fd701833",
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "{'product_desc': ['TP-Link TL-UE300 USB 3.0 to RJ45 Gigabit Ethernet Network Adapter Style name:UE300   UE300 adds gi
      "{'product_desc': ['TP-Link TL-UE300 USB 3.0 to RJ45 Gigabit Ethernet Network Adapter Style name:UE300   UE300 adds gi
      "{'product_desc': ['Maybelline New York Fit Me Matte with Poreless Foundation, 115 Ivory, 30ml Colour:115 Ivory   Its
     ]
    }
   ],
   "source": [
    "# convert data into dataset\n",
    "df = {'product_desc': data['product_description'], 'label': data['label']}\n",
    "from datasets import Dataset\n",
    "out_df = Dataset.from_dict(df)\n",
    "\n",
    "print(out_df[:2])\n",
    "\n",
    "#load a DistilBERT tokenizer, tokeniztation of the words\n",
    "from transformers import AutoTokenizer\n",
    "# using the distilbert tokenizer \n",
    "tokenizer = AutoTokenizer.from_pretrained(\"distilbert-base-uncased\", return_tensors='pt')\n",
    "\n",
    "#tokenization function\n",
    "# truncate sequences longer than DistilBERT?s maximum input length\n",
    "\n",
    "def tokenize(data):\n",
    "    return tokenizer(data[\"product_desc\"], truncation=True)\n",
    "\n",
    "#splitting dataset into train and test:\n",
    "train = out_df[:1000]\n",
    "test =  out_df[1000:1100]\n",
    "train = Dataset.from_dict(train)\n",
    "test = Dataset.from_dict(test)\n",
    "\n",
    "\n",
    "print(train[:2])\n",
    "print(test[:2])\n",
    "\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "f1b161bf",
   "metadata": {},
   "outputs": [],
   "source": [
    "print(test['label'])"
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "9edc1afe",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "('tokenizer\\\\tokenizer_config.json',\n",
       " 'tokenizer\\\\special_tokens_map.json',\n",
       " 'tokenizer\\\\vocab.txt',\n",
       " 'tokenizer\\\\added_tokens.json',\n",
       " 'tokenizer\\\\tokenizer.json')"
      ]
     },
     "execution_count": 3,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "#tokenizer = AutoTokenizer.from_pretrained(\"distilbert-base-uncased\")\n",
    "from transformers import AutoTokenizer, AutoModelForSequenceClassification\n",
    "tokenizer.save_pretrained(\"tokenizer\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "c166b6a7",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "08e9ece439b946859faa2aea4291d087",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "  0%|          | 0/1 [00:00<?, ?ba/s]"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    },
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "a6023076d538491e999bb46768d8d5a5",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "  0%|          | 0/1 [00:00<?, ?ba/s]"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "# tokenizing train and test sets using map function, to avoid creating tokenizers.Encoding objects and keep dict or Data
    "# objects with keys\n",
    "train_tokens = train.map(tokenize, batched=True)\n",
    "test_tokens= test.map(tokenize, batched=True)\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "c637f77b",
   "metadata": {},
   "outputs": [],
   "source": [
    "# datacollatorwithpadding: to create batches with padding to the maximum input length; pytorch implementation..\n",
    "from transformers import DataCollatorWithPadding\n",
    "data_collator = DataCollatorWithPadding(tokenizer=tokenizer)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 9,
   "id": "0bd50bcc",
   "metadata": {},
   "outputs": [],
   "source": [
    "# defining a function to calculate prediction performance of the model:  'accuracy' metric will be used\n",
    "\"\n",
    "import numpy as np\n",
    "import evaluate\n",
    "\"\n",
    "accuracy = evaluate.load(\"accuracy\")\n",
    "def acc_calculate(pred):\n",
    "    predicted, actual = pred\n",
    "    predicted = np.argmax(predicted, axis=1)# argmax returns indices of the maximum values along axis 1; \n",
    "    # predicted contains the probabilities of all classes for each entry in the dataset,\n",
    "    # the index 0 means that probabilty of class 0 is the highst and so on..\n",
    "    return accuracy.compute(predictions=predicted, references=actual)\n",
    "\"\n",
    "# create a map of the expected ids to their labels with id2label and label2id:\n",
```

```
    "\n",
    "id2label = {0: \"Electronics\", 1: \"Household\", 2: \"Books\", 3:\"Clothing & Accessories\"}\n",
    "label2id = {\"Electronics\": 0, \"Household\": 1, \"Books\":2, \"Clothing & Accessories\":3}\n",
    "\n",
    "## stop here if you are running the code on cpu; this will take around 90 Min. Use the pretraind model below.."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "id": "191fbb2c",
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "All TF 2.0 model weights were used when initializing DistilBertForSequenceClassification.\n",
      "\n",
      "All the weights of DistilBertForSequenceClassification were initialized from the TF 2.0 model.\n",
      "If your task is similar to the task the model of the checkpoint was trained on, you can already use DistilBertForSequ
     ]
    }
   ],
   "source": [
    "#model = AutoModelForSequenceClassification.from_pretrained(\"distilbert-base-uncased\", num_labels=4, id2label=id2label
    "#)\n",
    "#model.save_pretrained(\"distilbert_model_uncased\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 10,
   "id": "726ae769",
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "All TF 2.0 model weights were used when initializing DistilBertForSequenceClassification.\n",
      "\n",
      "All the weights of DistilBertForSequenceClassification were initialized from the TF 2.0 model.\n",
      "If your task is similar to the task the model of the checkpoint was trained on, you can already use DistilBertForSequ
     ]
    }
   ],
   "source": [
    "# defining the transformer\n",
    "#from_pretrained(): to load the pretrained Bert model weights\n",
    "#AutoModelForSequenceClassification: is a generic sequence classification model that will be instantiated using a preta
    "\n",
    "transformer = AutoModelForSequenceClassification.from_pretrained(\n",
    "\"distilbert-base-uncased\", num_labels=4, id2label=id2label, label2id=label2id, from_tf=True\n",
    ")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "7cea666f",
   "metadata": {},
   "outputs": [],
   "source": [
    "# defining the training parameters\n",
    "training_pars = TrainingArguments(\n",
    "    output_dir=\"my_transformer\",\n",
    "    per_device_train_batch_size=128,\n",
    "    per_device_eval_batch_size=128,\n",
    "    num_train_epochs=1,\n",
    "    weight_decay=0.01,\n",
    "    evaluation_strategy=\"epoch\",\n",
    "    save_strategy=\"epoch\",\n",
    "    load_best_model_at_end=True)\n",
    "\n",
    "# defining the trainer of the model\n",
    "#The Trainer class provides an API for feature-complete training in PyTorch\n",
    "trainer = Trainer(\n",
    "    model=transformer,\n",
    "    args=training_pars,\n",
    "    train_dataset= train_tokens,\n",
    "    eval_dataset=test_tokens,\n",
    "    tokenizer=tokenizer,\n",
    "    data_collator=data_collator,\n",
    "    compute_metrics=acc_calculate,\n",
    ")\n",
    "#training the transformer\n",
    "trainer.train()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "066486ac",
   "metadata": {
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "#save trainer to a file\n",
    "#trainer.save_model(\"./Distilbert_based_transformer\")\n",
    "\n",
    "# evaluate the model \n",
    "result =trainer.evaluate()\n",
```

```
    "result"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 11,
   "id": "c242a5c0",
   "metadata": {},
   "outputs": [],
   "source": [
    "## resume running the code from here using the saved model..\n",
    "transformer_trained = AutoModelForSequenceClassification.from_pretrained(\"./Distilbert_based_transformer\") "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 12,
   "id": "5863b1f7",
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "C:\\Users\\la2022\\anaconda3\\lib\\site-packages\\transformers\\pipelines\\text_classification.py:104: UserWarning: `
      "  warnings.warn(\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "[[{'label': 'Electronics', 'score': 0.9508970975875854},\n",
       "  {'label': 'Household', 'score': 0.017261911183595657},\n",
       "  {'label': 'Books', 'score': 0.017655229195952415},\n",
       "  {'label': 'Clothing & Accessories', 'score': 0.014185710810124874}]]"
      ]
     },
     "execution_count": 12,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "from transformers import TextClassificationPipeline\n",
    "# This pipeline has a return_all_scores parameter on its __call__ method that allows you to get all scores for each lab
    "pipe = TextClassificationPipeline(model=transformer_trained, tokenizer=tokenizer)\n",
    "prediction = pipe(\"NetGen 4k Wi-Fi 16 MP Ultra HD Action Camera\", return_all_scores=True)\n",
    "prediction"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "3a1c27f6",
   "metadata": {},
   "outputs": [],
   "source": [
    "# using GPT-2 to generate text\n",
    "from transformers import pipeline\n",
    "text_generator = pipeline('text-generation', model='gpt2')\n",
    "# generate 5 different sentences by sampling from the top 10 hits:\n",
    "#Temperature is a hyper-parameter used to control the randomness of predictions by scaling the logits before applying s
    "# when temperature is a large value (e.g. 1), the GPT-2 model produces more diversity and also more mistakes and viseve
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "b7f49a74",
   "metadata": {},
   "outputs": [],
   "source": [
    "#text_generator.save_pretrained('gpt2_pretrained')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "84513b65",
   "metadata": {},
   "outputs": [],
   "source": [
    "sentences = text_generator(\"in the last few years, a bunch of changes\", do_sample=True, top_k=10, temperature=0.6, ma
    "for sent in sentences:\n",
    "  print(sent[\"generated_text\"])\n",
    "  print(\"#\"*50)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "2db78df7",
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
```

```
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.13"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```


Distilbert_based_transformer.rar (248.37 MB)