```
# program: dropout_normalization_data_augmentation.py
# Description: Demonstrates dropout, batch normalization, and data augmentation in a simple CNN using Keras.

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess data
x_train = x_train.reshape((-1,28,28,1)).astype('float32') / 255.0
x_test = x_test.reshape((-1,28,28,1)).astype('float32') / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)
datagen.fit(x_train)

# Build a simple CNN model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.BatchNormalization(),        # Normalization
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),                # Dropout
```

```
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train with data augmentation
model.fit(datagen.flow(x_train, y_train, batch_size=64),
          epochs=5,
          validation_data=(x_test, y_test))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super
  self._warn_if_super_not_called()
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 103s 106ms/step - accuracy: 0.7562 - loss: 0.7937 - val_accuracy: 0.9811 - val_loss: 0.0597
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 140s 105ms/step - accuracy: 0.9398 - loss: 0.1996 - val_accuracy: 0.9880 - val_loss: 0.0350
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 97s 104ms/step - accuracy: 0.9538 - loss: 0.1470 - val_accuracy: 0.9831 - val_loss: 0.0532
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 142s 104ms/step - accuracy: 0.9605 - loss: 0.1289 - val_accuracy: 0.9889 - val_loss: 0.0322
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 99s 105ms/step - accuracy: 0.9667 - loss: 0.1101 - val_accuracy: 0.9899 - val_loss: 0.0290
<keras.src.callbacks.history.History at 0x7eac61920110>
```

Double-click (or enter) to edit

Start coding or generate with AI.

Start coding or generate with AI.

RNN Model

```
# rnn_model_example.py

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Generate dummy sequential data
# For example: 1000 samples, each with 10 time steps, each time step has 1 feature
X = np.random.randn(1000, 10, 1)
y = np.random.randn(1000, 1)

# Build RNN model
model = Sequential()
model.add(SimpleRNN(50, activation='tanh', input_shape=(10, 1)))
model.add(Dense(1))  # Output layer

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Summary
model.summary()

# Train the model
model.fit(X, y, epochs=20, batch_size=32)

# Make predictions
```

```
predictions = model.predict(X[:5])
print("Sample predictions:\n", predictions)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When u
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50)	2,600
dense (Dense)	(None, 1)	51

Total params: 2,651 (10.36 KB)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Data: 1000 samples, 10 time steps, 1 feature
X = np.random.randn(1000, 10, 1)
y = np.random.randn(1000, 1)

# Build RNN
model = Sequential()
model.add(SimpleRNN(50, activation='tanh', input_shape=(10,1)))
model.add(Dense(1))

# Compile
model.compile(optimizer='adam', loss='mse')

# Summary
model.summary()

# Train
model.fit(X, y, epochs=20, batch_size=32)

# Predict
print(model.predict(X[:5]))
```

```
Epoch 19/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.9491
```

```
Epoch 20/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.9280
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
Sample predictions:
 [[-0.40922505]
 [-0.00679891]
 [-0.37955353]
 [ 0.10581522]
 [-0.248231  ]]
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
simple_rnn_1 (SimpleRNN)	(None, 50)	2,600
dense_1 (Dense)	(None, 1)	51

```
 Total params: 2,651 (10.36 KB)
 Trainable params: 2,651 (10.36 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 1.1360
```

Double-click (or enter) to edit

Write the Python code for preprocessing text develop the RNN model with dense and embedding systemc indexing using Tensorflow.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Embedding
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 1.0348
```

```
texts = [
    "hello how are you",
    "I am fine thank you",
    "how about you",
    "I am doing well",
    "good to hear that"
]
```

```
Epoch 14/20
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

print("Word Index:", tokenizer.word_index)
print("Sequences:", sequences)

# Pad sequences so they all have the same length
```

```
X = pad_sequences(sequences, padding='post')
print("Padded Sequences:", X)
```

```
[ 0.25979906]
 [-0.15046753]]
Word Index: {'you': 1, 'how': 2, 'i': 3, 'am': 4, 'hello': 5, 'are': 6, 'fine': 7, 'thank': 8, 'about': 9, 'doing': 10, 'well': 11, 'good': 12, 'to': 13,
Sequences: [[5, 2, 6, 1], [3, 4, 7, 8, 1], [2, 9, 1], [3, 4, 10, 11], [12, 13, 14, 15]]
Padded Sequences: [[ 5  2  6  1  0]
 [ 3  4  7  8  1]
 [ 2  9  1  0  0]
 [ 3  4 10 11  0]
 [12 13 14 15  0]]
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

print("Word Index:", tokenizer.word_index)
print("Sequences:", sequences)

# Pad sequences so they all have the same length
X = pad_sequences(sequences, padding='post')
print("Padded Sequences:", X)
```

```
Word Index: {'you': 1, 'how': 2, 'i': 3, 'am': 4, 'hello': 5, 'are': 6, 'fine': 7, 'thank': 8, 'about': 9, 'doing': 10, 'well': 11, 'good': 12, 'to': 13,
Sequences: [[5, 2, 6, 1], [3, 4, 7, 8, 1], [2, 9, 1], [3, 4, 10, 11], [12, 13, 14, 15]]
Padded Sequences: [[ 5  2  6  1  0]
 [ 3  4  7  8  1]
 [ 2  9  1  0  0]
 [ 3  4 10 11  0]
 [12 13 14 15  0]]
```

```
vocab_size = len(tokenizer.word_index) + 1  # add 1 for padding token

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=8, input_length=X.shape[1]))
model.add(SimpleRNN(10))
model.add(Dense(1, activation='sigmoid'))  # example for binary output
```

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

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
simple_rnn_2 (SimpleRNN)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

```
# Example binary labels
y = np.array([1, 0, 1, 0, 1])

# Train
model.fit(X, y, epochs=10)
```

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.8000 - loss: 0.6818
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.8000 - loss: 0.6738
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - accuracy: 0.8000 - loss: 0.6658
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8000 - loss: 0.6579
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8000 - loss: 0.6499
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8000 - loss: 0.6419
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.8000 - loss: 0.6339
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.8000 - loss: 0.6259
Epoch 9/10
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8000 - loss: 0.6178
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8000 - loss: 0.6097
<keras.src.callbacks.history.History at 0x7a3289cbb950>
```

```
# Your new text prompt
new_text = ["good morning"]

# Convert to sequence
new_seq = tokenizer.texts_to_sequences(new_text)
new_padded = pad_sequences(new_seq, maxlen=X.shape[1], padding='post')

# Predict
prediction = model.predict(new_padded)
print("Prediction:", prediction)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Prediction: [[0.5908219]]
```

Bulid the LSTM Model using keras

```
import numpy as np
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

# Sample toy data: sequences of POS tag ids
# e.g. 0=NOUN, 1=VERB, 2=ADJ, etc.
X = [[0,1,2], [1,0,2], [0,2,1], [2,0,1]]
y = [1,2,0,1]  # next POS tag to predict

# Padding sequences
X = pad_sequences(X, maxlen=3, padding='pre')
y = to_categorical(y, num_classes=3)
```

```
# LSTM model
model = Sequential()
model.add(Embedding(input_dim=10, output_dim=8, input_length=3))
model.add(LSTM(16))
model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
model.fit(X, y, epochs=50, verbose=0)

# Prediction
test_seq = pad_sequences([[1,2,0]], maxlen=3, padding='pre')
pred = model.predict(test_seq)
print("Predicted class:", np.argmax(pred))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 194ms/step
Predicted class: 1
```

Start coding or generate with AI.

checking using LSTM and GRU uisng the Keras.

```
from keras.layers import GRU

# GRU model
model = Sequential()
model.add(Embedding(input_dim=10, output_dim=8, input_length=3))
model.add(GRU(16))
model.add(Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
model.fit(X, y, epochs=50, verbose=0)

# Prediction
pred = model.predict(test_seq)
print("Predicted class:", np.argmax(pred))
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
gru (GRU)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
1/1 ━━━━━━━━━━━━━━━━━━━━ **0s** 236ms/step
Predicted class: 1

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, GRU, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
# -------------------------
# DATA SETUP
# -------------------------
# Simulated sequences of word indices (like sentences)
X = [
    [1,2,3,4],    # example of grammatically correct sentence
    [2,3,0],      # example of grammatically incorrect sentence
    [5,6,7,8,9],  # correct
    [1,5,7],      # incorrect
    [2,4,8]       # correct
]

# Labels: 1=correct grammar, 0=incorrect grammar
y = [1, 0, 1, 0, 1]

# Pad sequences to same length
X = pad_sequences(X, maxlen=6, padding='post')

# -------------------------
# LSTM MODEL
# -------------------------
print("\n--------------------")
print("Training LSTM Model")
print("--------------------")
lstm_model = Sequential()
lstm_model.add(Embedding(input_dim=10, output_dim=8, input_length=6))
lstm_model.add(LSTM(16))
lstm_model.add(Dense(1, activation='sigmoid'))

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
lstm_model.summary()

# Train LSTM
lstm_model.fit(X, np.array(y), epochs=30, verbose=1)
```

```
# Predict with LSTM
test_sentence = pad_sequences([[2,3,4,5]], maxlen=6, padding='post')
lstm_pred = lstm_model.predict(test_sentence)
print("\nLSTM grammar probability (correct): {:.4f}".format(lstm_pred[0][0]))


# --------------------------
# GRU MODEL
# --------------------------
print("\n--------------------")
print("Training GRU Model")
print("--------------------")
gru_model = Sequential()
gru_model.add(Embedding(input_dim=10, output_dim=8, input_length=6))
gru_model.add(GRU(16))
gru_model.add(Dense(1, activation='sigmoid'))

gru_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
gru_model.summary()

# Train GRU
gru_model.fit(X, np.array(y), epochs=30, verbose=1)

# Predict with GRU
gru_pred = gru_model.predict(test_sentence)
print("\nGRU grammar probability (correct): {:.4f}".format(gru_pred[0][0]))
```

```
--------------------
Training LSTM Model
--------------------
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	?	0 (unbuilt)
lstm_1 (LSTM)	?	0 (unbuilt)
dense_3 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.6000 - loss: 0.6918
Epoch 2/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.6000 - loss: 0.6909
Epoch 3/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.6000 - loss: 0.6901
Epoch 4/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - accuracy: 0.6000 - loss: 0.6892
Epoch 5/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.6000 - loss: 0.6884
Epoch 6/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - accuracy: 0.6000 - loss: 0.6875
Epoch 7/30
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.6000 - loss: 0.6866
Epoch 8/30
```