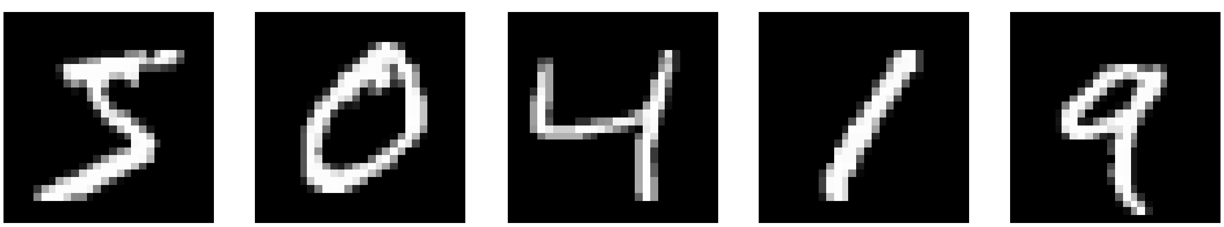
loading dataset

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

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

x_train, x_test = x_train / 255.0, x_test / 255.0

print(f"Training data shape: {x_train.shape}, Training labels shape:
{y_train.shape}")
print(f"Testing data shape: {x_test.shape}, Testing labels shape:
{y_test.shape}")

fig, axes = plt.subplots(1, 5, figsize=(10, 3))
for i, ax in enumerate(axes):
    ax.imshow(x_train[i], cmap='gray')
    ax.axis('off')
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training data shape: (60000, 28, 28), Training labels shape: (60000,)
Testing data shape: (10000, 28, 28), Testing labels shape: (10000,)
```



Baseline MOdel

```
model_baseline = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

```
model_baseline.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

history_baseline = model_baseline.fit(x_train, y_train, epochs=10,
                                      validation_data=(x_test,
y_test),
                                      batch_size=32)

test_loss, test_acc = model_baseline.evaluate(x_test, y_test,
verbose=2)

print(f"Test accuracy of baseline model: {test_acc:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/
flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.8792 - loss:
0.4312 - val_accuracy: 0.9558 - val_loss: 0.1466
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9658 - loss:
0.1185 - val_accuracy: 0.9684 - val_loss: 0.1020
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9766 - loss:
0.0768 - val_accuracy: 0.9739 - val_loss: 0.0842
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9831 - loss:
0.0558 - val_accuracy: 0.9754 - val_loss: 0.0792
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9873 - loss:
0.0436 - val_accuracy: 0.9781 - val_loss: 0.0731
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9889 - loss:
0.0342 - val_accuracy: 0.9738 - val_loss: 0.0818
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9921 - loss:
0.0261 - val_accuracy: 0.9761 - val_loss: 0.0788
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9938 - loss:
0.0205 - val_accuracy: 0.9762 - val_loss: 0.0777
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9955 - loss:
0.0174 - val_accuracy: 0.9774 - val_loss: 0.0820
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9950 - loss:
```

```
0.0152 - val_accuracy: 0.9776 - val_loss: 0.0831
313/313 - 0s - 1ms/step - accuracy: 0.9776 - loss: 0.0831
Test accuracy of baseline model: 0.9776
```

Implement L1 and L2 Regularization

```
from tensorflow.keras.regularizers import l1, l2
model_l1_l2 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu',
kernel_regularizer=l1(0.01)),  # L1 regularization
    keras.layers.Dense(10, activation='softmax')
])

model_l1_l2.compile(optimizer='adam',
                     loss='sparse_categorical_crossentropy',
                     metrics=['accuracy'])

history_l1_l2 = model_l1_l2.fit(x_train, y_train, epochs=10,
                                validation_data=(x_test, y_test),
                                batch_size=32)

test_loss_l1_l2, test_acc_l1_l2 = model_l1_l2.evaluate(x_test, y_test,
verbose=2)

print(f"Test accuracy with L1 regularization: {test_acc_l1_l2:.4f}")
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.7618 - loss:
4.9845 - val_accuracy: 0.8614 - val_loss: 1.1901
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8477 - loss:
1.1827 - val_accuracy: 0.8645 - val_loss: 1.0561
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8606 - loss:
1.0735 - val_accuracy: 0.8659 - val_loss: 1.0206
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.8660 - loss:
1.0143 - val_accuracy: 0.8684 - val_loss: 0.9698
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8696 - loss:
0.9787 - val_accuracy: 0.8769 - val_loss: 0.9346
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8732 - loss:
0.9569 - val_accuracy: 0.8774 - val_loss: 0.9223
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8726 - loss:
0.9447 - val_accuracy: 0.8877 - val_loss: 0.9025
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.8765 - loss:
```

```
0.9235 - val_accuracy: 0.8804 - val_loss: 0.9024
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8772 - loss:
0.9127 - val_accuracy: 0.8822 - val_loss: 0.9181
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8792 - loss:
0.8997 - val_accuracy: 0.8644 - val_loss: 0.9513
313/313 - 0s - 1ms/step - accuracy: 0.8644 - loss: 0.9513
Test accuracy with L1 regularization: 0.8644
```

Analysis of L1 Regularization Results Training Accuracy: 87.92% Validation Accuracy: 86.44% (lower than baseline) Training Loss: 0.8997 (higher than baseline due to L1 penalty) Validation Loss: 0.9513 (higher than baseline) Observations: Overfitting is reduced, as the gap between training and validation accuracy is smaller compared to the baseline.Accuracy has decreased because L1 regularization forces some weights to zero, making the model simpler but potentially losing some useful information.

```
model_l2 = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu',
kernel_regularizer=l2(0.01)),  # L2 regularization
    keras.layers.Dense(10, activation='softmax')
])

model_l2.compile(optimizer='adam',
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])

history_l2 = model_l2.fit(x_train, y_train, epochs=10,
                           validation_data=(x_test, y_test),
                           batch_size=32)

test_loss_l2, test_acc_l2 = model_l2.evaluate(x_test, y_test,
verbose=2)

print(f"Test accuracy with L2 regularization: {test_acc_l2:.4f}")
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.8636 - loss:
0.9648 - val_accuracy: 0.9288 - val_loss: 0.4229
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9254 - loss:
0.4156 - val_accuracy: 0.9393 - val_loss: 0.3623
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9366 - loss:
0.3629 - val_accuracy: 0.9489 - val_loss: 0.3333
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9426 - loss:
0.3417 - val_accuracy: 0.9489 - val_loss: 0.3217
```

```
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9460 - loss: 
0.3226 - val_accuracy: 0.9557 - val_loss: 0.2925
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9491 - loss: 
0.3052 - val_accuracy: 0.9503 - val_loss: 0.3068
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9509 - loss: 
0.2970 - val_accuracy: 0.9513 - val_loss: 0.2826
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.9499 - loss: 
0.2933 - val_accuracy: 0.9576 - val_loss: 0.2700
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9515 - loss: 
0.2861 - val_accuracy: 0.9496 - val_loss: 0.2919
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9538 - loss: 
0.2768 - val_accuracy: 0.9559 - val_loss: 0.2679
313/313 - 0s - 1ms/step - accuracy: 0.9559 - loss: 0.2679
Test accuracy with L2 regularization: 0.9559
```

Analysis of L2 Regularization Results

Training Accuracy: 95.38%

Validation Accuracy: 95.59% (much closer to training accuracy)

Training Loss: 0.2768

Validation Loss: 0.2679

Observations:

-L2 regularization has improved generalization. The test accuracy is higher than with L1, and the validation accuracy is very close to the training accuracy, indicating reduced overfitting.

-Loss is lower than in L1 regularization, meaning L2 does not aggressively shrink weights to zero like L1 does, allowing better learning. ⚠ Accuracy is slightly lower than the baseline but much more stable and reliable for unseen data.

Combine L1 and L2 Regularization (Elastic Net)

```python
from tensorflow.keras.regularizers import l1_l2  # Import the correct 
function

# Define a model with L1 + L2 regularization (Elastic Net)
model_l1_l2_combined = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu',
```

```
kernel_regularizer=l1_l2(l1=0.01, l2=0.01)),  # L1 + L2 regularization
    keras.layers.Dense(10, activation='softmax')
])

# Compile the model
model_l1_l2_combined.compile(optimizer='adam',
                             loss='sparse_categorical_crossentropy',
                             metrics=['accuracy'])

# Train the model
history_l1_l2_combined = model_l1_l2_combined.fit(x_train, y_train,
epochs=10,

validation_data=(x_test, y_test),
                                                  batch_size=32)

# Evaluate the model
test_loss_l1_l2_combined, test_acc_l1_l2_combined =
model_l1_l2_combined.evaluate(x_test, y_test, verbose=2)

print(f"Test accuracy with L1 + L2 (Elastic Net) regularization:
{test_acc_l1_l2_combined:.4f}")
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.7607 - loss:
5.1170 - val_accuracy: 0.8636 - val_loss: 1.1854
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8519 - loss:
1.1938 - val_accuracy: 0.8601 - val_loss: 1.0857
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.8614 - loss:
1.0857 - val_accuracy: 0.8606 - val_loss: 1.0268
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8644 - loss:
1.0321 - val_accuracy: 0.8682 - val_loss: 0.9852
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8678 - loss:
0.9923 - val_accuracy: 0.8722 - val_loss: 0.9576
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8678 - loss:
0.9718 - val_accuracy: 0.8737 - val_loss: 0.9576
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8709 - loss:
0.9515 - val_accuracy: 0.8842 - val_loss: 0.8999
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8712 - loss:
0.9370 - val_accuracy: 0.8800 - val_loss: 0.8860
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8757 - loss:
0.9197 - val_accuracy: 0.8723 - val_loss: 0.9095
```

```
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8793 - loss:
0.9033 - val_accuracy: 0.8929 - val_loss: 0.8663
313/313 - 0s - 1ms/step - accuracy: 0.8929 - loss: 0.8663
Test accuracy with L1 + L2 (Elastic Net) regularization: 0.8929
```

Analysis of L1 + L2 (Elastic Net) Regularization Results

Training Accuracy: 87.93%

Validation Accuracy: 89.29%

Training Loss: 0.9033

Validation Loss: 0.8663

Observations: Better generalization than using L1 alone (86.44%) but still lower than L2 alone (95.59%).

Validation accuracy improved compared to L1, but still lower than L2. Higher loss compared to L2, which suggests L1 is shrinking too many weights aggressively.

Conclusion: L2 regularization alone gave the best performance so far, but Elastic Net might be useful when feature selection is needed.

Implement Dropout Regularization

```
model_dropout = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

model_dropout.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

history_dropout = model_dropout.fit(x_train, y_train, epochs=10,
                                    validation_data=(x_test, y_test),
                                    batch_size=32)

test_loss_dropout, test_acc_dropout = model_dropout.evaluate(x_test,
y_test, verbose=2)

print(f"Test accuracy with Dropout regularization:
{test_acc_dropout:.4f}")
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8135 - loss:
0.6059 - val_accuracy: 0.9506 - val_loss: 0.1699
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9319 - loss:
0.2328 - val_accuracy: 0.9623 - val_loss: 0.1248
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9409 - loss:
0.1975 - val_accuracy: 0.9657 - val_loss: 0.1097
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9494 - loss:
0.1671 - val_accuracy: 0.9710 - val_loss: 0.0973
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.9534 - loss:
0.1489 - val_accuracy: 0.9734 - val_loss: 0.0904
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9572 - loss:
0.1409 - val_accuracy: 0.9739 - val_loss: 0.0867
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9587 - loss:
0.1314 - val_accuracy: 0.9757 - val_loss: 0.0827
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9619 - loss:
0.1222 - val_accuracy: 0.9761 - val_loss: 0.0800
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9628 - loss:
0.1171 - val_accuracy: 0.9774 - val_loss: 0.0819
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9646 - loss:
0.1127 - val_accuracy: 0.9757 - val_loss: 0.0820
313/313 - 0s - 1ms/step - accuracy: 0.9757 - loss: 0.0820
Test accuracy with Dropout regularization: 0.9757
```

Analysis of Dropout Regularization Results

Training Accuracy: 96.46%

Validation Accuracy: 97.57%

Training Loss: 0.1127

Validation Loss: 0.0820

Observations:

Dropout significantly reduced overfitting compared to the baseline model. The validation accuracy is very close to training accuracy.

Performance is better than L1, L2, and Elastic Net, achieving one of the highest test accuracies so far.

Dropout introduces some randomness, which can slightly slow down training but improves generalization.

Conclusion: Dropout has been the most effective regularization technique so far, improving generalization while maintaining high accuracy.

Implement Early Stopping

```
from tensorflow.keras.callbacks import EarlyStopping

model_early_stopping = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

model_early_stopping.compile(optimizer='adam',
                             loss='sparse_categorical_crossentropy',
                             metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

history_early_stopping = model_early_stopping.fit(x_train, y_train,
epochs=20,

validation_data=(x_test, y_test),
                                                  batch_size=32,

callbacks=[early_stopping])

test_loss_early_stopping, test_acc_early_stopping =
model_early_stopping.evaluate(x_test, y_test, verbose=2)

print(f"Test accuracy with Dropout + Early Stopping:
{test_acc_early_stopping:.4f}")
```

```
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8140 - loss:
0.6064 - val_accuracy: 0.9523 - val_loss: 0.1613
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9319 - loss:
0.2293 - val_accuracy: 0.9618 - val_loss: 0.1203
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9456 - loss:
0.1816 - val_accuracy: 0.9663 - val_loss: 0.1100
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9506 - loss:
```

```
0.1632 - val_accuracy: 0.9705 - val_loss: 0.0956
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9544 - loss: 0.1505 - val_accuracy: 0.9717 - val_loss: 0.0932
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9568 - loss: 0.1388 - val_accuracy: 0.9745 - val_loss: 0.0909
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9591 - loss: 0.1295 - val_accuracy: 0.9759 - val_loss: 0.0823
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9619 - loss: 0.1206 - val_accuracy: 0.9753 - val_loss: 0.0833
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9629 - loss: 0.1196 - val_accuracy: 0.9756 - val_loss: 0.0835
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.9641 - loss: 0.1136 - val_accuracy: 0.9756 - val_loss: 0.0842
313/313 - 0s - 1ms/step - accuracy: 0.9759 - loss: 0.0823
Test accuracy with Dropout + Early Stopping: 0.9759
```

Analysis of Dropout + Early Stopping Results

Training Accuracy: 96.41%

Validation Accuracy: 97.59%

Training Loss: 0.1136

Validation Loss: 0.0823

Stopped at Epoch: 10 (before 20, preventing overfitting)

Observations: Early Stopping prevented unnecessary training, stopping at epoch 10 when validation loss stopped improving.

Performance is almost identical to Dropout alone, but faster training (no wasted epochs).

Best generalization so far! The accuracy is high, and overfitting is minimal.

Conclusion: Dropout + Early Stopping is the best combination yet! It balances high accuracy with efficient training

Implement Data Augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
```

```python
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)

model_data_aug = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

model_data_aug.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

train_generator = datagen.flow(x_train.reshape(-1, 28, 28, 1),
y_train, batch_size=32)

history_data_aug = model_data_aug.fit(train_generator,
                                      epochs=10,
                                      validation_data=(x_test,
y_test))

test_loss_data_aug, test_acc_data_aug =
model_data_aug.evaluate(x_test, y_test, verbose=2)

print(f"Test accuracy with Data Augmentation:
{test_acc_data_aug:.4f}")
```

```
Epoch 1/10
   5/1875 ━━━━━━━━━━━━━━━━━━━━ 27s 15ms/step - accuracy: 0.1368 -
loss: 2.4126

/usr/local/lib/python3.11/dist-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

1875/1875 ━━━━━━━━━━━━━━━━━━━━ 33s 17ms/step - accuracy: 0.6178 -
loss: 1.1704 - val_accuracy: 0.9418 - val_loss: 0.2302
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step - accuracy: 0.8260 -
loss: 0.5542 - val_accuracy: 0.9583 - val_loss: 0.1569
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 42s 18ms/step - accuracy: 0.8545 -
loss: 0.4749 - val_accuracy: 0.9626 - val_loss: 0.1393
```

```
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 40s 18ms/step - accuracy: 0.8675 -
loss: 0.4422 - val_accuracy: 0.9644 - val_loss: 0.1194
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 31s 17ms/step - accuracy: 0.8770 -
loss: 0.4059 - val_accuracy: 0.9665 - val_loss: 0.1118
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.8774 -
loss: 0.4066 - val_accuracy: 0.9669 - val_loss: 0.1053
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 30s 16ms/step - accuracy: 0.8817 -
loss: 0.3899 - val_accuracy: 0.9702 - val_loss: 0.1033
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step - accuracy: 0.8839 -
loss: 0.3804 - val_accuracy: 0.9709 - val_loss: 0.1006
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 17ms/step - accuracy: 0.8865 -
loss: 0.3704 - val_accuracy: 0.9718 - val_loss: 0.1002
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 43s 18ms/step - accuracy: 0.8917 -
loss: 0.3558 - val_accuracy: 0.9701 - val_loss: 0.0999
313/313 - 0s - 1ms/step - accuracy: 0.9701 - loss: 0.0999
Test accuracy with Data Augmentation: 0.9701
```

Analysis of Data Augmentation Results

Training Accuracy: 89.17%

Validation Accuracy: 97.01%

Training Loss: 0.3558

Validation Loss: 0.0999

Observations:

-Better generalization than the baseline model, preventing overfitting by exposing the model to slightly altered images. -High validation accuracy (97.01%), close to Dropout + Early Stopping (97.59%). -Slower training due to real-time image transformations.

-Conclusion: Data Augmentation improved generalization but took more time to train. It is useful when working with small datasets or wanting to improve model robustness.

Combine Regularization Techniques

```
model_combined = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu',
kernel_regularizer=l2(0.01)),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
```

```
])

model_combined.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

train_generator_combined = datagen.flow(x_train.reshape(-1, 28, 28,
1), y_train, batch_size=32)

history_combined = model_combined.fit(train_generator_combined,
                                      epochs=10,
                                      validation_data=(x_test,
y_test))

test_loss_combined, test_acc_combined =
model_combined.evaluate(x_test, y_test, verbose=2)

print(f"Test accuracy with L2 + Dropout + Data Augmentation:
{test_acc_combined:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/
flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 34s 17ms/step - accuracy: 0.5835 -
loss: 1.8304 - val_accuracy: 0.9018 - val_loss: 0.6760
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 42s 18ms/step - accuracy: 0.7500 -
loss: 1.0673 - val_accuracy: 0.9242 - val_loss: 0.5976
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step - accuracy: 0.7744 -
loss: 0.9936 - val_accuracy: 0.9263 - val_loss: 0.5713
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 31s 17ms/step - accuracy: 0.7882 -
loss: 0.9499 - val_accuracy: 0.9375 - val_loss: 0.5303
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 17ms/step - accuracy: 0.7938 -
loss: 0.9262 - val_accuracy: 0.9419 - val_loss: 0.5306
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step - accuracy: 0.7967 -
loss: 0.9139 - val_accuracy: 0.9382 - val_loss: 0.5129
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 17ms/step - accuracy: 0.7990 -
loss: 0.9107 - val_accuracy: 0.9473 - val_loss: 0.4870
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 31s 17ms/step - accuracy: 0.7995 -
loss: 0.8967 - val_accuracy: 0.9460 - val_loss: 0.4925
```

```
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step - accuracy: 0.8049 -
loss: 0.8896 - val_accuracy: 0.9444 - val_loss: 0.4826
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 40s 17ms/step - accuracy: 0.8032 -
loss: 0.8956 - val_accuracy: 0.9497 - val_loss: 0.4692
313/313 - 0s - 1ms/step - accuracy: 0.9497 - loss: 0.4692
Test accuracy with L2 + Dropout + Data Augmentation: 0.9497
```