```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten,Conv2D,MaxPooling2D,Dropout,BatchNormalization
```

```
(X_train,y_train),(X_test,y_test)=keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
X_train.shape
```

```
(60000, 28, 28)
```

```
X_test.shape
```

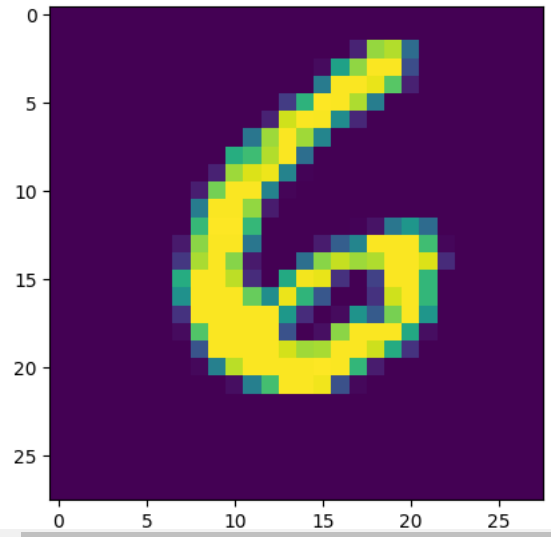
```
(10000, 28, 28)
```

```
y_test.shape
```

```
(10000,)
```

```
import matplotlib.pyplot as plt
plt.imshow(X_train[90])
```

```
<matplotlib.image.AxesImage at 0x7b20bbb09960>
```




```
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
```

```
X_train[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.        ],
[0.        ],
[0.        ]], dtype=float32)
```

```
X_train, X_val = X_train[:-10000], X_train[-10000:]
y_train, y_val = y_train[:-10000], y_train[-10000:]
```

```
X_train.shape
```

```
(50000, 28, 28, 1)
```

```
y_train
```

```
array([5, 0, 4, ..., 8, 4, 8], dtype=uint8)
```

```
# Build a simple CNN model with regularization
model = Sequential()

# Add layers to the model
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 64)	102,464
dense_1 (Dense)	(None, 10)	650

```
Total params: 121,930 (476.29 KB)
Trainable params: 121,930 (476.29 KB)
```

```
optimizer = tensorflow.keras.optimizers.Adam(learning_rate=0.001)
```

```
model.compile(loss = "sparse_categorical_crossentropy",optimizer = optimizer, metrics=['accuracy'])
```

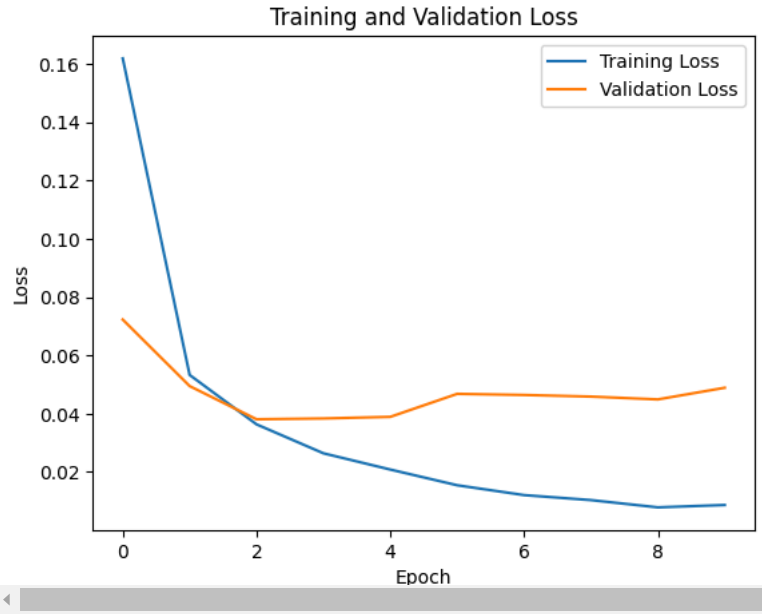
```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 40ms/step - accuracy: 0.8957 - loss: 0.3627 - val_accuracy: 0.9790 - val_loss: 0.0723
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 50s 32ms/step - accuracy: 0.9832 - loss: 0.0552 - val_accuracy: 0.9859 - val_loss: 0.0495
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 48s 31ms/step - accuracy: 0.9891 - loss: 0.0360 - val_accuracy: 0.9889 - val_loss: 0.0381
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 31ms/step - accuracy: 0.9916 - loss: 0.0247 - val_accuracy: 0.9896 - val_loss: 0.0384
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 31ms/step - accuracy: 0.9936 - loss: 0.0190 - val_accuracy: 0.9894 - val_loss: 0.0389
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 48s 31ms/step - accuracy: 0.9956 - loss: 0.0135 - val_accuracy: 0.9880 - val_loss: 0.0468
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 31ms/step - accuracy: 0.9960 - loss: 0.0108 - val_accuracy: 0.9885 - val_loss: 0.0464
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 47s 30ms/step - accuracy: 0.9974 - loss: 0.0081 - val_accuracy: 0.9894 - val_loss: 0.0459
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 87s 33ms/step - accuracy: 0.9972 - loss: 0.0077 - val_accuracy: 0.9888 - val_loss: 0.0449
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 47s 30ms/step - accuracy: 0.9976 - loss: 0.0071 - val_accuracy: 0.9894 - val_loss: 0.0489
```

```
#history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val), batch_size=32)

# Plotting the training and validation loss
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9895 - loss: 0.0402
Test Accuracy: 99.18%
```

Batch Noramlization and its usage:

Code after applying drop out layer

```
X_train
```

```
              [0.],
              ...,
              [0.],
              [0.],
              [0.]],

             [[0.],
              [0.],
              [0.],
              ...,
              [0.],
              [0.],
              [0.]]]], dtype=float32)
```

```
model_with_drop_out = Sequential()

# Input and first convolutional layer
model_with_drop_out.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_with_drop_out.add(MaxPooling2D((2, 2)))


model_with_drop_out.add(Dropout(0.25))

# Second convolutional layer
model_with_drop_out.add(Conv2D(64, (3, 3), activation='relu'))
model_with_drop_out.add(MaxPooling2D((2, 2)))

model_with_drop_out.add(Dropout(0.25))

model_with_drop_out.add(Flatten())


model_with_drop_out.add(Dense(64, activation='relu'))
model_with_drop_out.add(Dropout(0.5))


model_with_drop_out.add(Dense(10, activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
optimizer = tensorflow.keras.optimizers.Adam(learning_rate=0.001)
```

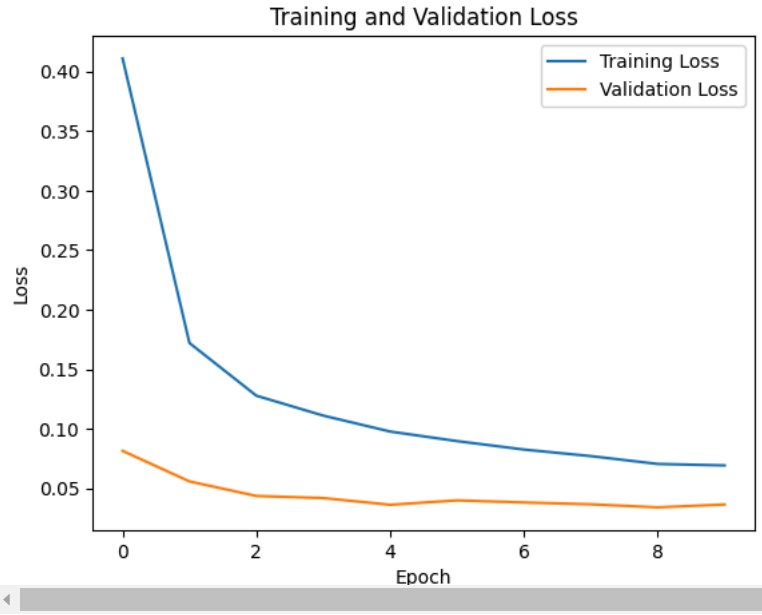
```
model_with_drop_out.compile(loss = "sparse_categorical_crossentropy",optimizer = optimizer, metrics=['accuracy'])
```

```
history_with_dropout = model_with_drop_out.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 64s 40ms/step - accuracy: 0.7611 - loss: 0.7232 - val_accuracy: 0.9759 - val_loss: 0.0815
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 55s 35ms/step - accuracy: 0.9425 - loss: 0.1841 - val_accuracy: 0.9832 - val_loss: 0.0559
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 35ms/step - accuracy: 0.9616 - loss: 0.1337 - val_accuracy: 0.9876 - val_loss: 0.0437
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 35ms/step - accuracy: 0.9654 - loss: 0.1131 - val_accuracy: 0.9879 - val_loss: 0.0419
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 35ms/step - accuracy: 0.9689 - loss: 0.0972 - val_accuracy: 0.9897 - val_loss: 0.0363
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 54s 35ms/step - accuracy: 0.9735 - loss: 0.0851 - val_accuracy: 0.9897 - val_loss: 0.0400
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 54s 35ms/step - accuracy: 0.9747 - loss: 0.0823 - val_accuracy: 0.9902 - val_loss: 0.0382
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 54s 35ms/step - accuracy: 0.9767 - loss: 0.0777 - val_accuracy: 0.9905 - val_loss: 0.0367
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 35ms/step - accuracy: 0.9792 - loss: 0.0692 - val_accuracy: 0.9914 - val_loss: 0.0341
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 54s 35ms/step - accuracy: 0.9799 - loss: 0.0671 - val_accuracy: 0.9902 - val_loss: 0.0365
```

```
import matplotlib.pyplot as plt

plt.plot(history_with_dropout.history['loss'], label='Training Loss')
plt.plot(history_with_dropout.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss with drop out layer')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
test_loss, test_accuracy = model_with_drop_out.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.9899 - loss: 0.0323
Test Accuracy: 99.17%
```

```
model_with_BN = Sequential()

# Add layers to the model
model_with_BN.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model_with_BN.add(BatchNormalization())
model_with_BN.add(MaxPooling2D((2, 2)))
model_with_BN.add(Conv2D(64, (3, 3), activation='relu'))
model_with_BN.add(BatchNormalization())
model_with_BN.add(MaxPooling2D((2, 2)))
model_with_BN.add(Flatten())
model_with_BN.add(Dense(64, activation='relu'))
model_with_BN.add(BatchNormalization())
model_with_BN.add(Dense(10, activation='softmax'))
```

```
optimizer = tensorflow.keras.optimizers.Adam(learning_rate=0.001)
```

```
model_with_BN.compile(loss = "sparse_categorical_crossentropy",optimizer = optimizer, metrics=['accuracy'])
```

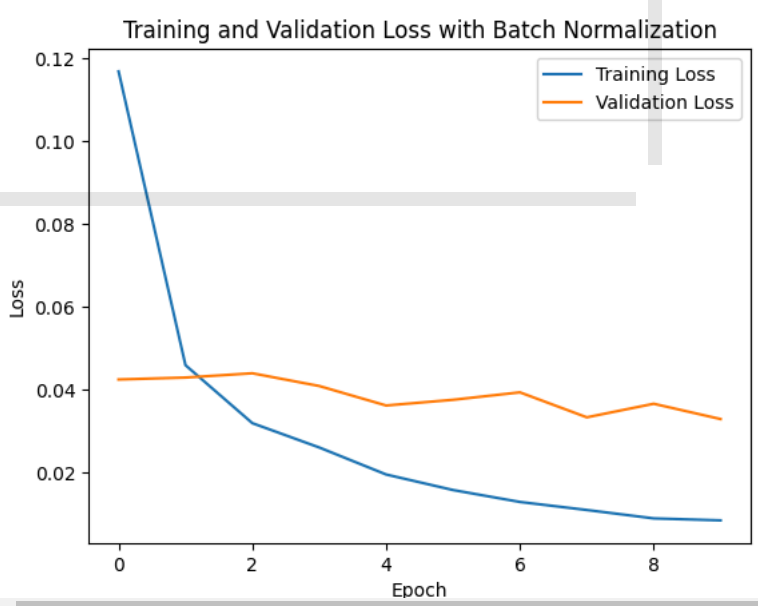
```
history_with_BN = model_with_BN.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 91s 56ms/step - accuracy: 0.9294 - loss: 0.2394 - val_accuracy: 0.9877 - val_loss: 0.0425
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 70s 45ms/step - accuracy: 0.9879 - loss: 0.0431 - val_accuracy: 0.9873 - val_loss: 0.0430
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 44ms/step - accuracy: 0.9911 - loss: 0.0291 - val_accuracy: 0.9876 - val_loss: 0.0440
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - accuracy: 0.9924 - loss: 0.0226 - val_accuracy: 0.9889 - val_loss: 0.0410
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 45ms/step - accuracy: 0.9943 - loss: 0.0189 - val_accuracy: 0.9892 - val_loss: 0.0363
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - accuracy: 0.9952 - loss: 0.0147 - val_accuracy: 0.9894 - val_loss: 0.0376
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - accuracy: 0.9964 - loss: 0.0114 - val_accuracy: 0.9902 - val_loss: 0.0394
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 46ms/step - accuracy: 0.9972 - loss: 0.0095 - val_accuracy: 0.9916 - val_loss: 0.0334
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 79s 44ms/step - accuracy: 0.9975 - loss: 0.0079 - val_accuracy: 0.9902 - val_loss: 0.0367
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 44ms/step - accuracy: 0.9980 - loss: 0.0064 - val_accuracy: 0.9908 - val_loss: 0.0330
```

```
import matplotlib.pyplot as plt

plt.plot(history_with_BN.history['loss'], label='Training Loss')
plt.plot(history_with_BN.history['val_loss'], label='Validation Loss'
plt.title('Training and Validation Loss with Batch Normalization')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.legend()
```




```
test_loss, test_accuracy = model_with_BN.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy*100:.2f}%")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9893 - loss: 0.0361
Test Accuracy: 99.10%
```

Double-click (or enter) to edit

Batch Noramlization and its usage:

Batch normalization is a technique which makes deep learning training more faster and stable. It consists of normalizing activation vectors from hidden layers using the mean and variance of the current batch. This is applied right before or after the nonlinear function

Drop out layer and its usage:

Dropout layer is a technique used to reduce overfitting of the model by dropping of some neurons randomly from hidden layer to avoid overfitting

```
Inferences:
```

I have trained the model without dropout layer, later with dropout layer and batch normalization layer.