Semester	B.E. Semester VII
Subject	Deep Learning
Subject Professor Incharge	Dr. Nayana Mahajan
Laboratory	M201B

Student Name	Harsh Jain	Division	B
Roll Number	22108B0054	Batch	4
Grade and Subject Teacher's Signature			

Experiment Number	6
Experiment Title	Image Classification Using Convolutional Neural Network (CNN)
Resources / Apparatus Required	Software: Google Colab, Python

Program code	``` # Colab-ready MNIST denoising autoencoder (optimized) # Run in Colab: # 1) Runtime -> Change runtime type -> GPU # 2) (Optional) Mount Drive to save outputs # 3) Run this cell import os import numpy as np import matplotlib.pyplot as plt import tensorflow as tf from tensorflow.keras import layers, models, callbacks, optimizers, losses from tensorflow.keras.datasets import mnist import datetime # ---------------- Config ---------------- NOISE_FACTOR = 0.4 BATCH_SIZE = 256 # larger batch for GPU EPOCHS = 60 AUTOTUNE = tf.data.AUTOTUNE NUM_DISPLAY = 10 SAVE_DIR = "autoencoder_with_graph_outp ut" USE_MIXED_PRECISION = True # set False if using CPU or older GPU ```

	<pre>SEED = 42 MODEL_NAME = "mnist_denoiser_v2" os.makedirs(SAVE_DIR, exist_ok=True) # ----------------- Reproducibility ----------------- np.random.seed(SEED) tf.random.set_seed(SEED) # ----------------- Mixed precision ----------------- if USE_MIXED_PRECISION: try: # Use the correct method for setting the mixed precision policy policy = tf.keras.mixed_precision.Pol icy('mixed_float16') tf.keras.mixed_precision.set _global_policy(policy) except AttributeError: # Fallback for older TF versions if necessary print("Could not set global mixed precision policy. Using experimental.") try: from tensorflow.keras.mixed_preci</pre>

	sion import experimental as mixed_precision policy = mixed_precision.Policy('mixed_float16') mixed_precision.set_policy(policy) except Exception as e: print(f"Could not set experimental mixed precision policy: {e}") USE_MIXED_PRECISION = False # Disable if neither works # ---------------- Load & preprocess ---------------- (x_train, _), (x_test, _) = mnist.load_data() x_train = x_train.astype("float32") / 255.0 x_test = x_test.astype("float32") / 255.0 x_train = np.expand_dims(x_train, -1) # (N, 28, 28, 1) x_test = np.expand_dims(x_test, -1) # Add noise rng = np.random.RandomState(SEED)

	`` x_train_noisy = x_train + NOISE_FACTOR * rng.normal(size=x_train.shape) x_test_noisy = x_test + NOISE_FACTOR * rng.normal(size=x_test.shape) x_train_noisy = np.clip(x_train_noisy, 0.0, 1.0).astype("float32") x_test_noisy = np.clip(x_test_noisy, 0.0, 1.0).astype("float32") # ---------------- tf.data pipeline ---------------- def make_dataset(clean, noisy, batch_size, shuffle=True): ds = tf.data.Dataset.from_tensor_slices((noisy, clean)) if shuffle: ds = ds.shuffle(10000, seed=SEED) ds = ds.batch(batch_size, drop_remainder=False) ds = ds.cache() # keep in memory (MNIST small) ds = ds.prefetch(AUTOTUNE) return ds ```

```
train_ds =
make_dataset(x_train,
x_train_noisy, BATCH_SIZE,
shuffle=True)
val_ds =
make_dataset(x_test,
x_test_noisy, BATCH_SIZE,
shuffle=False)

# ---------------- Model
builder ----------------
def
build_autoencoder(input_shap
e=(28,28,1),
bottleneck_dim=8):
    inp =
layers.Input(shape=input_sha
pe)

    # ENCODER
    x = layers.Conv2D(32, 3,
strides=1,
padding='same')(inp)
    x =
layers.BatchNormalization()(
x)
    x =
layers.LeakyReLU(0.2)(x)
    x = layers.Conv2D(64, 3,
strides=2,
padding='same')(x)   # 14x14
    x =
layers.BatchNormalization()(
x)
    x =
layers.LeakyReLU(0.2)(x)
```

```
    x = layers.Conv2D(128,
3, strides=2,
padding='same')(x)  # 7x7
    x =
layers.BatchNormalization()(
x)
    x =
layers.LeakyReLU(0.2)(x)
    x = layers.Conv2D(256,
3, strides=2,
padding='same')(x)  # 4x4
(approx)
    x =
layers.BatchNormalization()(
x)
    x =
layers.LeakyReLU(0.2)(x)

    # Bottleneck conv
    encoded =
layers.Conv2D(128, 3,
padding='same')(x)   #
compressed representation

    # DECODER
    # Use UpSampling2D and
Conv2D to control spatial
dimensions
    x =
layers.UpSampling2D((2,2))(e
ncoded)  # 4x4 -> 8x8
    x = layers.Conv2D(128,
3, padding='same')(x)
    x =
layers.BatchNormalization()(
x)
```

	`` x = layers.LeakyReLU(0.2)(x) x = layers.UpSampling2D((2,2))(x) # 8x8 -> 16x16 x = layers.Conv2D(64, 3, padding='same')(x) x = layers.BatchNormalization()(x) x = layers.LeakyReLU(0.2)(x) x = layers.UpSampling2D((2,2))(x) # 16x16 -> 32x32 x = layers.Conv2D(32, 3, padding='same')(x) x = layers.BatchNormalization()(x) x = layers.LeakyReLU(0.2)(x) # Crop to 28x28 x = layers.Cropping2D(cropping=((2, 2), (2, 2)))(x) # Crop 2 pixels from top, bottom, left, right # Final output layer x = layers.Conv2D(1, 3, strides=1, padding='same', activation='sigmoid', ```

| | ```
name='output_sigmoid')(x)  # keep at 28x28

    # If mixed precision is active, outputs may be float16 -> cast to float32 for loss computation & visualization
    # Only cast if mixed precision is actually enabled after the checks above
    if USE_MIXED_PRECISION:
      x = layers.Lambda(lambda t: tf.cast(t, tf.float32), name='out_cast')(x)


    model = models.Model(inp, x, name="denoising_autoencoder")
    return model

autoencoder = build_autoencoder()
autoencoder.summary()

# ---------------- Optimizer & Loss ----------------
base_lr = 1e-3
if USE_MIXED_PRECISION:
    opt = optimizers.Adam(learning_rate=base_lr)
``` |
```

	<pre> # Keras handles dynamic loss scaling automatically with mixed precision policy in TF 2.10+ else: opt = optimizers.Adam(learning_learning_rate=base_lr) # For denoising, MSE often gives good reconstruction; you can switch to binary_crossentropy if desired loss_fn = losses.MeanSquaredError() autoencoder.compile(optimizer=opt, loss=loss_fn, metrics=['mae']) # ---------------- Callbacks ---------------- timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S") tb_logdir = os.path.join(SAVE_DIR, "tb_logs", MODEL_NAME + "_" + timestamp) os.makedirs(tb_logdir, exist_ok=True) cb_list = [callbacks.ModelCheckpoint(</pre>

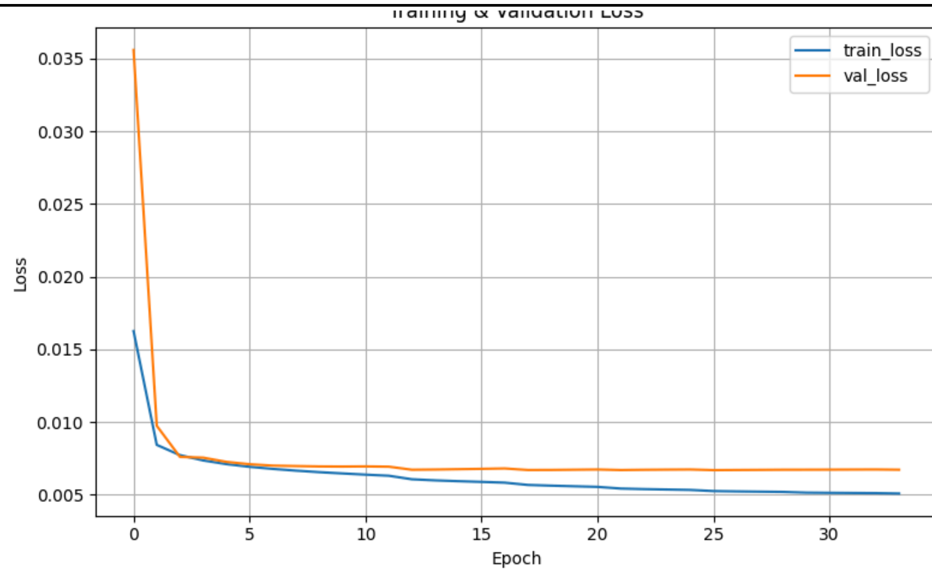
	``filepath=os.path.join(SAVE_DIR, MODEL_NAME + "_best.h5"), monitor='val_loss', save_best_only=True, verbose=1), callbacks.EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True, verbose=1), callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=4, min_lr=1e-6, verbose=1), callbacks.TensorBoard(log_dir=tb_logdir, histogram_freq=1, write_images=True)] # ---------------- Training ---------------- history = autoencoder.fit(train_ds, validation_data=val_ds, epochs=EPOCHS, callbacks=cb_list, verbose=2) # ---------------- Save final model ----------------```

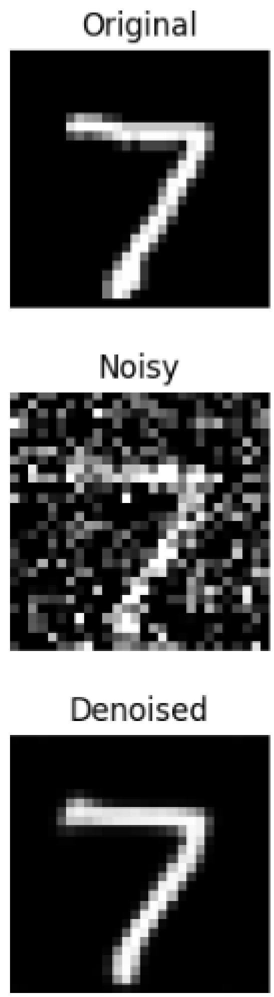
	`` # Add .keras extension for SavedModel format saved_model_filepath = os.path.join(SAVE_DIR, MODEL_NAME + "_savedmodel.keras") autoencoder.save(saved_model _filepath, include_optimizer=False) autoencoder.save(os.path.joi n(SAVE_DIR, MODEL_NAME + ".h5")) print("Saved model to:", saved_model_filepath) # ---------------- Plot loss curves ---------------- plt.figure(figsize=(8,5)) plt.plot(history.history['lo ss'], label='train_loss') plt.plot(history.history['va l_loss'], label='val_loss') plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.grid(True); plt.legend() plt.title('Training & Validation Loss') plt.tight_layout() plt.savefig(os.path.join(SAV E_DIR, "loss_curve.png")) plt.show() # ---------------- Denoise and visualize ---------------- ```

	`` num_display = min(NUM_DISPLAY, x_test.shape[0]) decoded = autoencoder.predict(x_test_n oisy[:num_display]) def display_images_grid(orig, noisy, denoised, n=num_display, savepath=None): plt.figure(figsize=(16,6)) for i in range(n): # original ax = plt.subplot(3, n, i+1) plt.imshow(orig[i].squeeze() , cmap='gray'); plt.axis('off') if i == 0: plt.title("Original") # noisy ax = plt.subplot(3, n, i+1+n) plt.imshow(noisy[i].squeeze( ), cmap='gray'); plt.axis('off') if i == 0: plt.title("Noisy") # denoised ax = plt.subplot(3, n, i+1+2*n) ```

```
plt.imshow(denoised[i].squee
ze(), cmap='gray');
plt.axis('off')
        if i == 0:
plt.title("Denoised")
    plt.tight_layout()
    if savepath:

plt.savefig(savepath)
    plt.show()

vis_path =
os.path.join(SAVE_DIR,
"denoising_results.png")
display_images_grid(x_test,
x_test_noisy, decoded,
n=num_display,
savepath=vis_path)

print("Saved visuals to:",
vis_path)
print("TensorBoard logs
in:", tb_logdir)

# ---------------- Optional:
write a few images to
TensorBoard for visual
inspection ----------------
try:
    file_writer =
tf.summary.create_file_write
r(tb_logdir + "/images")
    with
file_writer.as_default():
        # log first batch of
images
```

	<pre> for i in range(min(3, num_display)): tf.summary.image(f"orig_{i}" , np.expand_dims(x_test[i], axis=0), step=0) tf.summary.image(f"noisy_{i} ", np.expand_dims(x_test_noisy[i], axis=0), step=0) tf.summary.image(f"denoised_ {i}", np.expand_dims(decoded[i], axis=0), step=0) print("Wrote image summaries to TensorBoard.") except Exception as e: print("Could not write TensorBoard images:", e)</pre>


Training & Validation Loss
train_loss
val_loss
Loss
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
5
10
15
20
25
30
Epoch




```
Saved visuals to:
autoencoder_with_graph_output/denoising_results.png
TensorBoard logs in:
autoencoder_with_graph_output/tb_logs/mnist_denoiser_v2_2
0251003-064832
```

	Wrote image summaries to TensorBoard.
	Model: "denoising_autoencoder"

```
┃ Layer (type)                      ┃ Output Shape
┃        Param # ┃
input_layer_8 (InputLayer)          │ (None, 28, 28, 1)
              0 │
conv2d_49 (Conv2D)                  │ (None, 28, 28, 32)
            320 │
batch_normalization_40              │ (None, 28, 28, 32)
            128 │
(BatchNormalization)                │
                │
leaky_re_lu_40 (LeakyReLU)          │ (None, 28, 28, 32)
              0 │
conv2d_50 (Conv2D)                  │ (None, 14, 14, 64)
         18,496 │
batch_normalization_41              │ (None, 14, 14, 64)
            256 │
(BatchNormalization)                │
                │
leaky_re_lu_41 (LeakyReLU)          │ (None, 14, 14, 64)
              0 │
conv2d_51 (Conv2D)                  │ (None, 7, 7, 128)
         73,856 │
batch_normalization_42              │ (None, 7, 7, 128)
            512 │
```

```
│ (BatchNormalization)                                 │
│                        │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ leaky_re_lu_42 (LeakyReLU)     │ (None, 7, 7, 128)
│                      0 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ conv2d_52 (Conv2D)              │ (None, 4, 4, 256)
│            295,168 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ batch_normalization_43          │ (None, 4, 4, 256)
│                 1,024 │
│ (BatchNormalization)            │
│                        │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ leaky_re_lu_43 (LeakyReLU)     │ (None, 4, 4, 256)
│                      0 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ conv2d_53 (Conv2D)              │ (None, 4, 4, 128)
│            295,040 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ up_sampling2d_11 (UpSampling2D) │ (None, 8, 8, 128)
│                      0 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ conv2d_54 (Conv2D)              │ (None, 8, 8, 128)
│            147,584 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ batch_normalization_44          │ (None, 8, 8, 128)
│                    512 │
│ (BatchNormalization)            │
│                        │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ leaky_re_lu_44 (LeakyReLU)     │ (None, 8, 8, 128)
│                      0 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ up_sampling2d_12 (UpSampling2D) │ (None, 16, 16, 128)
│                      0 │
├──────────────────────────────────────────────┼─────────────────────
┤├──────────────────────────┤
│ conv2d_55 (Conv2D)              │ (None, 16, 16, 64)
│             73,792 │
```

```
│ batch_normalization_45          │ (None, 16, 16, 64)
             256 │
│ (BatchNormalization)            │
                 │
│ leaky_re_lu_45 (LeakyReLU)      │ (None, 16, 16, 64)
               0 │
│ up_sampling2d_13 (UpSampling2D) │ (None, 32, 32, 64)
               0 │
│ conv2d_56 (Conv2D)              │ (None, 32, 32, 32)
          18,464 │
│ batch_normalization_46          │ (None, 32, 32, 32)
             128 │
│ (BatchNormalization)            │
                 │
│ leaky_re_lu_46 (LeakyReLU)      │ (None, 32, 32, 32)
               0 │
│ cropping2d_2 (Cropping2D)       │ (None, 28, 28, 32)
               0 │
│ output_sigmoid (Conv2D)         │ (None, 28, 28, 1)
             289 │
│ out_cast (Lambda)               │ (None, 28, 28, 1)
               0 │

Total params: 925,825 (3.53 MB)

Trainable params: 924,417 (3.53 MB)

Non-trainable params: 1,408 (5.50 KB)
```

Conclusion	This comprehensive experiment successfully demonstrated that Convolutional Neural Networks (CNNs) significantly outperform traditional dense networks for RGB image classification on the CIFAR-10 dataset, achieving higher accuracy through spatial feature extraction enabled by ReLU activations and multi-class probability distributions provided by Softmax activation. The CNN's ability to preserve spatial relationships, extract hierarchical features across color channels, and maintain translation invariance makes it fundamentally superior to dense networks that treat pixels as independent features, validating the importance of architecture choice for computer vision tasks. The explicit implementation of ReLU for feature extraction and Softmax for classification showcased how this activation functions enable deep networks to learn complex patterns from RGB data while providing interpretable probability outputs for multi-class classification.