|  |  |
| --- | --- |
| Semester | B.E. Semester VII |
| Subject | Deep Learning |
| Subject Professor In-charge | Dr. Nayana Mahajan |
| Laboratory | M201B |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student Name | Harsh Jian | Division | | B |
| Roll Number | 22108B0054 | Batch | | 4 |
| Grade and Subject Teacher’s Signature |  | |  | |

|  |  |
| --- | --- |
| Experiment Number | 4 |
| Experiment Title |  |
| Resources / Apparatus Required | Software: |
| Algorithm |  |
| Program code | import tensorflow as tf  from tensorflow.keras import datasets, layers, models  import matplotlib.pyplot as plt  # ========== Dataset Loaders ==========  (x\_train\_mnist, y\_train\_mnist), (x\_test\_mnist, y\_test\_mnist) = datasets.mnist.load\_data()  x\_train\_mnist, x\_test\_mnist = x\_train\_mnist / 255.0, x\_test\_mnist / 255.0  x\_train\_mnist, x\_test\_mnist = tf.image.resize(x\_train\_mnist[..., tf.newaxis], [32,32]), tf.image.resize(x\_test\_mnist[..., tf.newaxis], [32,32])  (x\_train\_cifar, y\_train\_cifar), (x\_test\_cifar, y\_test\_cifar) = datasets.cifar10.load\_data()  x\_train\_cifar, x\_test\_cifar = x\_train\_cifar / 255.0, x\_test\_cifar / 255.0  # ========== LeNet on MNIST ==========  def build\_lenet():      return models.Sequential([          layers.Conv2D(6, (5,5), activation='relu', input\_shape=(32,32,1)),          layers.AveragePooling2D((2,2)),          layers.Conv2D(16, (5,5), activation='relu'),          layers.AveragePooling2D((2,2)),          layers.Flatten(),          layers.Dense(120, activation='relu'),          layers.Dense(84, activation='relu'),          layers.Dense(10, activation='softmax')      ])  lenet = build\_lenet()  lenet.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  history\_lenet = lenet.fit(x\_train\_mnist, y\_train\_mnist, epochs=5, batch\_size=128, validation\_data=(x\_test\_mnist, y\_test\_mnist), verbose=1)  # ========== AlexNet on CIFAR-10 ==========  def build\_alexnet():      return models.Sequential([          # Adjusted kernel size and strides for 32x32 input          layers.Conv2D(96, (5,5), strides=1, activation='relu', input\_shape=(32,32,3), padding="same"),          layers.MaxPooling2D((2,2), strides=2),          layers.Conv2D(256, (3,3), activation='relu', padding="same"),          layers.MaxPooling2D((2,2), strides=2),          layers.Conv2D(384, (3,3), activation='relu', padding="same"),          layers.Conv2D(384, (3,3), activation='relu', padding="same"),          layers.Conv2D(256, (3,3), activation='relu', padding="same"),          layers.MaxPooling2D((2,2), strides=2), # Adjusted stride          layers.Flatten(),          layers.Dense(4096, activation='relu'),          layers.Dropout(0.5),          layers.Dense(4096, activation='relu'),          layers.Dropout(0.5),          layers.Dense(10, activation='softmax')      ])  alexnet = build\_alexnet()  alexnet.compile(optimizer=tf.keras.optimizers.SGD(learning\_rate=0.01, momentum=0.9), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  history\_alexnet = alexnet.fit(x\_train\_cifar, y\_train\_cifar, epochs=5, batch\_size=128, validation\_data=(x\_test\_cifar, y\_test\_cifar), verbose=1)  # ========== ResNet, DenseNet, EfficientNet (transfer learning) ==========  from tensorflow.keras.applications import ResNet50, DenseNet121, EfficientNetB0  from tensorflow.keras.applications.resnet50 import preprocess\_input as resnet\_pre  from tensorflow.keras.applications.densenet import preprocess\_input as dense\_pre  from tensorflow.keras.applications.efficientnet import preprocess\_input as eff\_pre  def build\_transfer\_model(base, preprocess\_func, input\_shape=(32,32,3), num\_classes=10):      base\_model = base(weights=None, include\_top=False, input\_shape=input\_shape)      model = models.Sequential([          layers.Lambda(preprocess\_func, input\_shape=input\_shape),          base\_model,          layers.Flatten(),          layers.Dense(128, activation='relu'),          layers.Dense(num\_classes, activation='softmax')      ])      return model  resnet = build\_transfer\_model(ResNet50, resnet\_pre)  resnet.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  history\_resnet = resnet.fit(x\_train\_cifar, y\_train\_cifar, epochs=5, batch\_size=128, validation\_data=(x\_test\_cifar, y\_test\_cifar), verbose=1)  densenet = build\_transfer\_model(DenseNet121, dense\_pre)  densenet.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  history\_densenet = densenet.fit(x\_train\_cifar, y\_train\_cifar, epochs=5, batch\_size=128, validation\_data=(x\_test\_cifar, y\_test\_cifar), verbose=1)  efficientnet = build\_transfer\_model(EfficientNetB0, eff\_pre)  efficientnet.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  history\_efficientnet = efficientnet.fit(x\_train\_cifar, y\_train\_cifar, epochs=5, batch\_size=128, validation\_data=(x\_test\_cifar, y\_test\_cifar), verbose=1)  # ========== Plot Accuracy Comparison ==========  plt.figure(figsize=(12,6))  plt.plot(history\_lenet.history['val\_accuracy'], label="LeNet-MNIST")  plt.plot(history\_alexnet.history['val\_accuracy'], label="AlexNet-CIFAR10")  plt.plot(history\_resnet.history['val\_accuracy'], label="ResNet-CIFAR10")  plt.plot(history\_densenet.history['val\_accuracy'], label="DenseNet-CIFAR10")  plt.plot(history\_efficientnet.history['val\_accuracy'], label="EfficientNet-CIFAR10")  plt.legend(); plt.xlabel("Epochs"); plt.ylabel("Validation Accuracy"); plt.title("Model Comparison")  plt.show() |
| Output |  |
|  | /usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base\_conv.py:113: 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\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)  Epoch 1/5  **469/469** ━━━━━━━━━━━━━━━━━━━━ **8s** 10ms/step - accuracy: 0.7893 - loss: 0.7103 - val\_accuracy: 0.9640 - val\_loss: 0.1095  Epoch 2/5  **469/469** ━━━━━━━━━━━━━━━━━━━━ **5s** 4ms/step - accuracy: 0.9670 - loss: 0.1070 - val\_accuracy: 0.9783 - val\_loss: 0.0726  Epoch 3/5  **469/469** ━━━━━━━━━━━━━━━━━━━━ **2s** 4ms/step - accuracy: 0.9773 - loss: 0.0742 - val\_accuracy: 0.9812 - val\_loss: 0.0612  Epoch 4/5  **469/469** ━━━━━━━━━━━━━━━━━━━━ **2s** 4ms/step - accuracy: 0.9827 - loss: 0.0587 - val\_accuracy: 0.9840 - val\_loss: 0.0490  Epoch 5/5  **469/469** ━━━━━━━━━━━━━━━━━━━━ **3s** 4ms/step - accuracy: 0.9855 - loss: 0.0490 - val\_accuracy: 0.9807 - val\_loss: 0.0574  Epoch 1/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **39s** 78ms/step - accuracy: 0.1789 - loss: 2.1623 - val\_accuracy: 0.3748 - val\_loss: 1.7111  Epoch 2/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **24s** 53ms/step - accuracy: 0.3961 - loss: 1.6283 - val\_accuracy: 0.5216 - val\_loss: 1.3262  Epoch 3/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **21s** 54ms/step - accuracy: 0.5264 - loss: 1.3068 - val\_accuracy: 0.6015 - val\_loss: 1.1069  Epoch 4/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **40s** 52ms/step - accuracy: 0.6085 - loss: 1.0979 - val\_accuracy: 0.6451 - val\_loss: 1.0128  Epoch 5/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **21s** 53ms/step - accuracy: 0.6712 - loss: 0.9333 - val\_accuracy: 0.6889 - val\_loss: 0.8986  Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5>  **94765736/94765736** ━━━━━━━━━━━━━━━━━━━━ **5s** 0us/step  /usr/local/lib/python3.12/dist-packages/keras/src/layers/core/lambda\_layer.py:65: 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 1/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **101s** 123ms/step - accuracy: 0.2890 - loss: 1.9624 - val\_accuracy: 0.1000 - val\_loss: 3.8138  Epoch 2/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **30s** 48ms/step - accuracy: 0.5227 - loss: 1.3482 - val\_accuracy: 0.1000 - val\_loss: 63.6282  Epoch 3/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 48ms/step - accuracy: 0.5927 - loss: 1.1673 - val\_accuracy: 0.1000 - val\_loss: 34.3242  Epoch 4/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 48ms/step - accuracy: 0.6167 - loss: 1.0922 - val\_accuracy: 0.1000 - val\_loss: 53.3736  Epoch 5/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **21s** 48ms/step - accuracy: 0.6395 - loss: 1.0237 - val\_accuracy: 0.1000 - val\_loss: 122.2200  Downloading data from <https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5>  **29084464/29084464** ━━━━━━━━━━━━━━━━━━━━ **2s** 0us/step  Epoch 1/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **282s** 337ms/step - accuracy: 0.3400 - loss: 1.8086 - val\_accuracy: 0.1000 - val\_loss: 12.3270  Epoch 2/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 52ms/step - accuracy: 0.5333 - loss: 1.3078 - val\_accuracy: 0.1000 - val\_loss: 12.8014  Epoch 3/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **21s** 54ms/step - accuracy: 0.6070 - loss: 1.1049 - val\_accuracy: 0.1000 - val\_loss: 12.9312  Epoch 4/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **40s** 52ms/step - accuracy: 0.6362 - loss: 1.0356 - val\_accuracy: 0.1119 - val\_loss: 30.1140  Epoch 5/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 51ms/step - accuracy: 0.6555 - loss: 0.9751 - val\_accuracy: 0.1000 - val\_loss: 68.1657  Downloading data from <https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5>  **16705208/16705208** ━━━━━━━━━━━━━━━━━━━━ **2s** 0us/step  Epoch 1/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **155s** 200ms/step - accuracy: 0.4258 - loss: 1.6306 - val\_accuracy: 0.1011 - val\_loss: 2.6275  Epoch 2/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **32s** 27ms/step - accuracy: 0.6987 - loss: 0.8602 - val\_accuracy: 0.1157 - val\_loss: 4.2127  Epoch 3/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 26ms/step - accuracy: 0.7766 - loss: 0.6478 - val\_accuracy: 0.1496 - val\_loss: 2.8199  Epoch 4/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 26ms/step - accuracy: 0.8099 - loss: 0.5444 - val\_accuracy: 0.1061 - val\_loss: 2.6136  Epoch 5/5  **391/391** ━━━━━━━━━━━━━━━━━━━━ **20s** 26ms/step - accuracy: 0.8438 - loss: 0.4537 - val\_accuracy: 0.1010 - val\_loss: 4.2113 |
| Conclusion |  |