**1. Deep Learning.**

**a. Build a DNN with five hidden layers of 100 neurons each, He initialization, and the**

**ELU activation function.**

**b. Using Adam optimization and early stopping, try training it on MNIST but only on**

**digits 0 to 4, as we will use transfer learning for digits 5 to 9 in the next exercise. You**

**will need a softmax output layer with five neurons, and as always make sure to save**

**checkpoints at regular intervals and save the final model so you can reuse it later.**

**c. Tune the hyperparameters using cross-validation and see what precision you can**

**achieve.**

**d. Now try adding Batch Normalization and compare the learning curves: is it**

**converging faster than before? Does it produce a better model?**

**e. Is the model overfitting the training set? Try adding dropout to every layer and try**

**again. Does it help?**

**Answer:**

(a) Build a DNN with 5 hidden layers of 100 neurons each

* Use He initialization for weight initialization.
* Use the ELU activation function.

import tensorflow as tf

from tensorflow import keras

# Define the model

model = keras.models.Sequential([

keras.layers.Flatten(input\_shape=[28, 28]), # Input layer

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(5, activation="softmax") # Output layer for 5 classes (0-4)

])

model.summary()

**(b) Train with Adam, Early Stopping, and Checkpoints**

* Use **Adam optimizer**.
* Implement **early stopping**.
* Save **model checkpoints**.

# Load MNIST dataset and filter digits 0-4

(X\_train\_full, y\_train\_full), (X\_test, y\_test) = keras.datasets.mnist.load\_data()

X\_train, y\_train = X\_train\_full[y\_train\_full < 5], y\_train\_full[y\_train\_full < 5]

X\_test, y\_test = X\_test[y\_test < 5], y\_test[y\_test < 5]

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0 # Normalize

# Callbacks

checkpoint\_cb = keras.callbacks.ModelCheckpoint("mnist\_dnn.h5", save\_best\_only=True)

early\_stopping\_cb = keras.callbacks.EarlyStopping(patience=5, restore\_best\_weights=True)

# Compile and train

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

history = model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test),

callbacks=[checkpoint\_cb, early\_stopping\_cb])

**(c) Hyperparameter Tuning**

* Use **KerasTuner** for hyperparameter tuning.

import keras\_tuner as kt

def build\_model(hp):

model = keras.models.Sequential([

keras.layers.Flatten(input\_shape=[28, 28]),

keras.layers.Dense(hp.Int("units", 50, 200, step=50), activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(hp.Int("units", 50, 200, step=50), activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(5, activation="softmax")

])

model.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

return model

tuner = kt.RandomSearch(build\_model, objective="val\_accuracy", max\_trials=5, directory="mnist\_tuning")

tuner.search(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

best\_hps = tuner.get\_best\_hyperparameters(num\_trials=1)[0]

**(d) Add Batch Normalization and Compare**

* **Batch normalization** speeds up training and improves stability.

model\_bn = keras.models.Sequential([

keras.layers.Flatten(input\_shape=[28, 28]),

keras.layers.Dense(100, kernel\_initializer="he\_normal"),

keras.layers.BatchNormalization(),

keras.layers.Activation("elu"),

keras.layers.Dense(100, kernel\_initializer="he\_normal"),

keras.layers.BatchNormalization(),

keras.layers.Activation("elu"),

keras.layers.Dense(100, kernel\_initializer="he\_normal"),

keras.layers.BatchNormalization(),

keras.layers.Activation("elu"),

keras.layers.Dense(5, activation="softmax")

])

model\_bn.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

history\_bn = model\_bn.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test),

callbacks=[checkpoint\_cb, early\_stopping\_cb])

**(e) Add Dropout to Prevent Overfitting**

* **Dropout** prevents overfitting by randomly dropping neurons.

model\_dropout = keras.models.Sequential([

keras.layers.Flatten(input\_shape=[28, 28]),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dropout(0.2),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dropout(0.2),

keras.layers.Dense(5, activation="softmax")

])

model\_dropout.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

history\_dropout = model\_dropout.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test),

callbacks=[checkpoint\_cb, early\_stopping\_cb])

**2. Transfer learning.**

**a. Create a new DNN that reuses all the pretrained hidden layers of the previous**

**model, freezes them, and replaces the softmax output layer with a new one.**

**b. Train this new DNN on digits 5 to 9, using only 100 images per digit, and time how**

**long it takes. Despite this small number of examples, can you achieve high precision?**

**c. Try caching the frozen layers, and train the model again: how much faster is it now?**

**d. Try again reusing just four hidden layers instead of five. Can you achieve a higher**

**precision?**

**e. Now unfreeze the top two hidden layers and continue training: can you get the**

**model to perform even better?**

**Answer:**

1. **Reuse pretrained hidden layers, freeze them, and replace the output layer**

base\_model = keras.models.load\_model("mnist\_dnn.h5")

for layer in base\_model.layers[:-1]:

layer.trainable = False # Freeze all hidden layers

model\_transfer = keras.models.Sequential(base\_model.layers[:-1])

model\_transfer.add(keras.layers.Dense(5, activation="softmax")) # New output layer for digits 5-9

1. **Train on digits 5-9 with only 100 images per digit**

X\_train, y\_train = X\_train\_full[y\_train\_full >= 5][:500], y\_train\_full[y\_train\_full >= 5][:500]

X\_test, y\_test = X\_test[y\_test >= 5], y\_test[y\_test >= 5]

model\_transfer.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

model\_transfer.fit(X\_train, y\_train - 5, epochs=30, validation\_data=(X\_test, y\_test - 5),

callbacks=[checkpoint\_cb, early\_stopping\_cb])

**3. Pretraining on an auxiliary task.**

**a. In this exercise you will build a DNN that compares two MNIST digit images and**

**predicts whether they represent the same digit or not. Then you will reuse the lower**

**layers of this network to train an MNIST classifier using very little training data. Start**

**by building two DNNs (let’s call them DNN A and B), both similar to the one you built**

**earlier but without the output layer: each DNN should have five hidden layers of 100**

**neurons each, He initialization, and ELU activation. Next, add one more hidden layer**

**with 10 units on top of both DNNs. To do this, you should use**

**TensorFlow’s concat() function with axis=1 to concatenate the outputs of both DNNs**

**for each instance, then feed the result to the hidden layer. Finally, add an output**

**layer with a single neuron using the logistic activation function.**

**b. Split the MNIST training set in two sets: split #1 should containing 55,000 images,**

**and split #2 should contain contain 5,000 images. Create a function that generates a**

**training batch where each instance is a pair of MNIST images picked from split #1.**

**Half of the training instances should be pairs of images that belong to the same**

**class, while the other half should be images from different classes. For each pair, the**

**training label should be 0 if the images are from the same class, or 1 if they are from**

**different classes.**

**c. Train the DNN on this training set. For each image pair, you can simultaneously feed**

**the first image to DNN A and the second image to DNN B. The whole network will**

**gradually learn to tell whether two images belong to the same class or not.**

**d. Now create a new DNN by reusing and freezing the hidden layers of DNN A and**

**adding a softmax output layer on top with 10 neurons. Train this network on split #2**

**and see if you can achieve high performance despite having only 500 images per**

**class.**

**Answer:**

1. Siamese Network for Image Similarity

input\_A = keras.layers.Input(shape=[28, 28])

input\_B = keras.layers.Input(shape=[28, 28])

shared\_DNN = keras.models.Sequential([

keras.layers.Flatten(),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

keras.layers.Dense(100, activation="elu", kernel\_initializer="he\_normal"),

])

output\_A = shared\_DNN(input\_A)

output\_B = shared\_DNN(input\_B)

merged = keras.layers.concatenate([output\_A, output\_B])

hidden = keras.layers.Dense(10, activation="elu")(merged)

output = keras.layers.Dense(1, activation="sigmoid")(hidden)

siamese\_model = keras.models.Model(inputs=[input\_A, input\_B], outputs=output)

1. **Create Pairs of Similar/Dissimilar Images**

import numpy as np

def generate\_pairs(X, y):

pairs, labels = [], []

for i in range(len(X)):

idx = np.random.choice(np.where(y == y[i])[0]) # Same class

pairs.append([X[i], X[idx]])

labels.append(0)

idx = np.random.choice(np.where(y != y[i])[0]) # Different class

pairs.append([X[i], X[idx]])

labels.append(1)

return np.array(pairs), np.array(labels)

pairs, labels = generate\_pairs(X\_train, y\_train)

1. **Train the Siamese Network**

siamese\_model.compile(loss="binary\_crossentropy", optimizer="adam", metrics=["accuracy"])

siamese\_model.fit([pairs[:, 0], pairs[:, 1]], labels, epochs=10, batch\_size=32, validation\_split=0.1)

1. **Transfer Lower Layers to MNIST Classifier**

new\_classifier = keras.models.Sequential(siamese\_model.layers[:-1]) # Remove output layer

new\_classifier.add(keras.layers.Dense(10, activation="softmax")) # MNIST classification

new\_classifier.compile(loss="sparse\_categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

new\_classifier.fit(X\_train, y\_train, epochs=30, validation\_data=(X\_test, y\_test))