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?

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

**import tensorflow as tf**

**from tensorflow import keras**

**model = keras.models.Sequential()**

**model.add(keras.layers.InputLayer(input\_shape=(784,))) # Assuming MNIST images are flattened to 784 features.**

**for \_ in range(5):**

**model.add(keras.layers.Dense(100, activation='elu', kernel\_initializer='he\_normal'))**

**model.add(keras.layers.Dense(5, activation='softmax')) # Output layer for digits 0 to 4.**

**b. Train the model on MNIST digits 0 to 4 using Adam optimization, early stopping, and checkpoint saving.**

**c. Tune hyperparameters using cross-validation to achieve the best precision.**

**d. Add Batch Normalization layers and compare the learning curves. You can use keras.layers.BatchNormalization() after each Dense layer.**

**e. Address overfitting by adding dropout layers. You can use keras.layers.Dropout(rate) after each Dense layer.**

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?

1. **Create a new DNN that reuses pretrained hidden layers, freezes them, and replaces the softmax output layer.**

**# Load the previously trained model**

**pretrained\_model = keras.models.load\_model("pretrained\_model.h5")**

**# Freeze pretrained layers**

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

**layer.trainable = False**

**# Create a new output layer for digits 5 to 9**

**new\_output\_layer = keras.layers.Dense(5, activation='softmax')**

**# Build the new DNN**

**new\_model = keras.models.Sequential(pretrained\_model.layers[:-1] + [new\_output\_layer])**

**b. Train the new DNN on digits 5 to 9 using a limited number of images per digit.**

**c. Cache the frozen layers for faster training. This can be done by setting trainable to True for the cached layers.**

**d. Experiment with different configurations, such as using four hidden layers instead of five.**

**e. Unfreeze the top two hidden layers and continue training to see if model performance improves.**

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.

1. **Build two DNNs (DNN A and B) with shared layers up to the last hidden layer, and add an output layer for binary classification.**

**# Create DNN A and DNN B up to the shared last hidden layer**

**shared\_layers = [keras.layers.InputLayer(input\_shape=(784,))] # Assuming MNIST images are flattened to 784 features.**

**for \_ in range(5):**

**shared\_layers.append(keras.layers.Dense(100, activation='elu', kernel\_initializer='he\_normal'))**

**shared\_output = keras.layers.Concatenate(axis=1)([shared\_layer.output for shared\_layer in shared\_layers[1:]])**

**shared\_output = keras.layers.Dense(10, activation='elu', kernel\_initializer='he\_normal')(shared\_output)**

**# Create DNN A**

**dnn\_a = keras.models.Sequential(shared\_layers + [keras.layers.Dense(1, activation='sigmoid')])**

**# Create DNN B**

**dnn\_b = keras.models.Sequential(shared\_layers + [keras.layers.Dense(1, activation='sigmoid')])**

**b. Create a training batch generator that generates image pairs and labels as described in split #1.**

**c. Train DNN A and DNN B on the training batch.**

**d. Create a new DNN by reusing and freezing the shared layers of DNN A and adding a softmax output layer for 10 classes.**

**# Load the pretrained shared layers**

**pretrained\_shared\_layers = keras.models.load\_model("shared\_layers.h5", custom\_objects={'tf': tf})**

**# Freeze the shared layers**

**for layer in pretrained\_shared\_layers.layers:**

**layer.trainable = False**

**# Create a new output layer for 10 classes**

**new\_output\_layer = keras.layers.Dense(10, activation='softmax')**

**# Build the new DNN**

**new\_dnn = keras.models.Sequential([pretrained\_shared\_layers] + [new\_output\_layer])**

**e. Train the new DNN on split #2 to achieve high performance using limited data.**