exercise_1

November 9, 2025

1 Exercise-1: Training Deep Neural Network on MNIST

Train a controlled deep neural network on the MNIST dataset. Set random seeds to 42. Load and preprocess MNIST. Build the network using the following configuration: - Flatten input images to $28 \times 28 = 784$ features - 3 hidden layers, 64 neurons each - ELU activation function - He normal initialization - Output layer: 10 neurons with softmax - Optimizer: Nadam - learning_rate = 0.001, loss=sparse_categorical_crossentropy - EarlyStopping callback: monitor validation loss, patience = 5, restore best weights - epochs = 50, batch size = 32 - Use only the first 1000 training samples and first 200 test samples

1.1 Q1.1 Report the obtained test accuracy.

[2]:
```
# Load MNIST dataset
from tensorflow.keras.datasets import mnist

RANDOM_SEED = 42
hidden_layers = 3
hidden_neurons = 64
output_neurons = 10
learning_rate = 0.001

# Load data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess data, normalize and reshape
x_train = x_train.reshape(-1, 28 * 28).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28 * 28).astype('float32') / 255.0

# Use only the first 1000 training samples and first 200 test samples
x_train = x_train[:1000]
y_train = y_train[:1000]
x_test = x_test[:200]
y_test = y_test[:200]

import tensorflow as tf
tf.random.set_seed(RANDOM_SEED)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

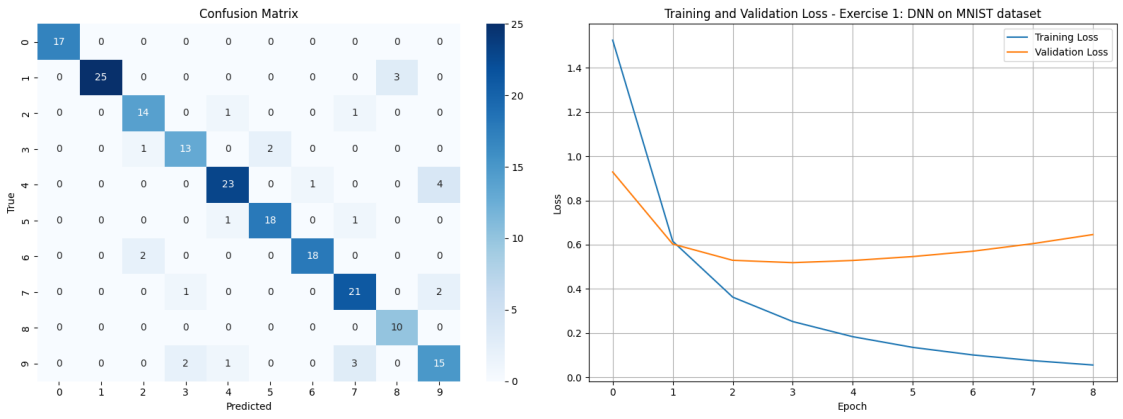
```
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.callbacks import EarlyStopping

# Build the model
model = Sequential()
# Input layer
model.add(Dense(hidden_neurons, activation='elu',␣
 ↪kernel_initializer='he_normal', input_shape=(28 * 28,)))
# Hidden layers, 3 layers with ELU activation and He normal initialization
for _ in range(hidden_layers - 1):
    model.add(Dense(hidden_neurons, activation='elu',␣
 ↪kernel_initializer='he_normal'))
# Output layer with softmax activation
model.add(Dense(output_neurons, activation='softmax'))
model.compile(optimizer=Nadam(learning_rate=learning_rate),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


# Early stopping callback
earl = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# Train the model
history = model.fit(x_train,
                    y_train,
                    epochs=50,
                    batch_size=32,
                    validation_split=0.2,
                    callbacks=[earl],
                    verbose=1)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Epoch 1/50

/opt/miniconda3/envs/envAssignments/lib/python3.13/site-
packages/keras/src/layers/core/dense.py:92: 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)

25/25             1s 6ms/step -
accuracy: 0.5337 - loss: 1.5253 - val_accuracy: 0.7400 - val_loss: 0.9290
Epoch 2/50
25/25             0s 1ms/step -
accuracy: 0.8475 - loss: 0.6158 - val_accuracy: 0.8250 - val_loss: 0.6030
Epoch 3/50
```

```
25/25             0s 1ms/step -
accuracy: 0.9062 - loss: 0.3629 - val_accuracy: 0.8450 - val_loss: 0.5294
Epoch 4/50
25/25             0s 1ms/step -
accuracy: 0.9438 - loss: 0.2524 - val_accuracy: 0.8750 - val_loss: 0.5188
Epoch 5/50
25/25             0s 1ms/step -
accuracy: 0.9613 - loss: 0.1839 - val_accuracy: 0.8600 - val_loss: 0.5287
Epoch 6/50
25/25             0s 2ms/step -
accuracy: 0.9775 - loss: 0.1356 - val_accuracy: 0.8400 - val_loss: 0.5464
Epoch 7/50
25/25             0s 1ms/step -
accuracy: 0.9862 - loss: 0.1013 - val_accuracy: 0.8400 - val_loss: 0.5706
Epoch 8/50
25/25             0s 1ms/step -
accuracy: 0.9912 - loss: 0.0757 - val_accuracy: 0.8400 - val_loss: 0.6049
Epoch 9/50
25/25             0s 2ms/step -
accuracy: 0.9937 - loss: 0.0558 - val_accuracy: 0.8300 - val_loss: 0.6459
Test Accuracy: 87.00%
```

[3]:
```
from helper import plot_model_evaluation

# Plot model evaluation
plot_model_evaluation(model, x_test, y_test, history, exercise_title='Exercise␣
↪1: DNN on MNIST dataset')
```

```
7/7               0s 4ms/step
```



[ ]:

exercise_2

November 9, 2025

1 Exercise-2: Training Deep Neural Network on CIFAR-10

Train a controlled deep neural network on the CIFAR-10 dataset. Set random seeds to 42. Load and preprocess CIFAR-10. Build the network using the following configuration: - Flatten input images to $32 \times 32 \times 3 = 3072$ features - 4 hidden layers, 256 neurons each - ELU activation function - He normal initialization - Output layer: 10 neurons with softmax - Optimizer: Nadam - learning_rate = 0.001, loss = sparse_categorical_crossentropy - EarlyStopping callback: monitor validation loss, patience = 5, restore best weights - epochs = 50, batch_size = 128 - Use only the first 5000 training samples and first 1000 test samples

1.1 Q2.1 Report the obtained test accuracy.

[2]:
```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.layers import Dense
from tensorflow.random import set_seed
import matplotlib.pyplot as plt
import numpy as np

loss_function = 'sparse_categorical_crossentropy'
output_activation = 'softmax'
initializer = 'he_normal'
learning_rate = 0.001
hidden_neurons = 256
output_neurons = 10
optimizer = 'Nadam'
activation = 'elu'
random_seed = 42
batch_size = 128
epochs = 50

# Set random seeds to 42
set_seed(42)
np.random.seed(42)

# Load cifar-10 dataset
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Use only the first 5000 training samples and first 1000 test samples
x_train = x_train[:5000]
y_train = y_train[:5000]
x_test = x_test[:1000]
y_test = y_test[:1000]

# Preprocess the data by flattening and normalizing, 32x32x3 to 3072 features
x_train = x_train.reshape(-1, 32 * 32 * 3).astype('float32') / 255.0
x_test = x_test.reshape(-1, 32 * 32 * 3).astype('float32') / 255.0

# Build the model
model = Sequential()
# Add hidden layers
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer,
                        input_shape=(32 * 32 * 3,)))
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer))
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer))
model.add(Dense(hidden_neurons,
                        activation=activation,
                        kernel_initializer=initializer))

# add output layer
model.add(Dense(output_neurons, activation=output_activation))
# Compile the model, with specified optimizer and loss function
model.compile(optimizer=Nadam(learning_rate=learning_rate),
              loss=loss_function,
              metrics=['accuracy'])

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss',
                       patience=5,
                       restore_best_weights=True)
# Train the model
history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2,
          callbacks=[early_stopping],
          verbose=1)
```

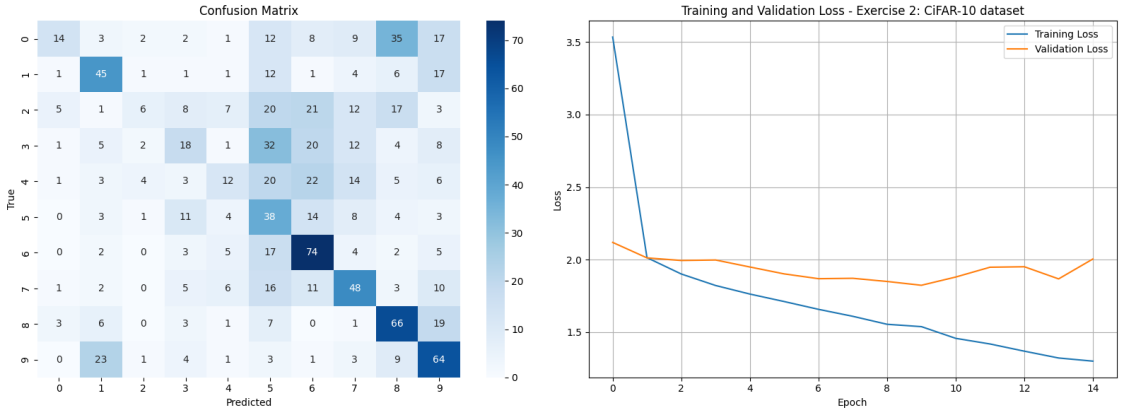
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
Epoch 1/50
/opt/miniconda3/envs/envAssignments/lib/python3.13/site-
packages/keras/src/layers/core/dense.py:92: 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)
32/32             1s 8ms/step -
accuracy: 0.1570 - loss: 3.5349 - val_accuracy: 0.2200 - val_loss: 2.1185
Epoch 2/50
32/32             0s 5ms/step -
accuracy: 0.2718 - loss: 2.0133 - val_accuracy: 0.2590 - val_loss: 2.0119
Epoch 3/50
32/32             0s 5ms/step -
accuracy: 0.3207 - loss: 1.9006 - val_accuracy: 0.2780 - val_loss: 1.9938
Epoch 4/50
32/32             0s 5ms/step -
accuracy: 0.3485 - loss: 1.8213 - val_accuracy: 0.2780 - val_loss: 1.9972
Epoch 5/50
32/32             0s 6ms/step -
accuracy: 0.3730 - loss: 1.7623 - val_accuracy: 0.2990 - val_loss: 1.9490
Epoch 6/50
32/32             0s 5ms/step -
accuracy: 0.3868 - loss: 1.7107 - val_accuracy: 0.3290 - val_loss: 1.9014
Epoch 7/50
32/32             0s 6ms/step -
accuracy: 0.4075 - loss: 1.6569 - val_accuracy: 0.3600 - val_loss: 1.8683
Epoch 8/50
32/32             0s 6ms/step -
accuracy: 0.4263 - loss: 1.6088 - val_accuracy: 0.3530 - val_loss: 1.8710
Epoch 9/50
32/32             0s 6ms/step -
accuracy: 0.4462 - loss: 1.5538 - val_accuracy: 0.3740 - val_loss: 1.8491
Epoch 10/50
32/32             0s 6ms/step -
accuracy: 0.4523 - loss: 1.5376 - val_accuracy: 0.3860 - val_loss: 1.8229
Epoch 11/50
32/32             0s 6ms/step -
accuracy: 0.4745 - loss: 1.4568 - val_accuracy: 0.3730 - val_loss: 1.8797
Epoch 12/50
32/32             0s 7ms/step -
accuracy: 0.4945 - loss: 1.4180 - val_accuracy: 0.3630 - val_loss: 1.9474
Epoch 13/50
32/32             0s 6ms/step -
```

```
accuracy: 0.5130 - loss: 1.3683 - val_accuracy: 0.3690 - val_loss: 1.9507
Epoch 14/50
32/32             0s 6ms/step -
accuracy: 0.5360 - loss: 1.3212 - val_accuracy: 0.3980 - val_loss: 1.8669
Epoch 15/50
32/32             0s 6ms/step -
accuracy: 0.5355 - loss: 1.2996 - val_accuracy: 0.3540 - val_loss: 2.0042
32/32             0s 926us/step -
accuracy: 0.3850 - loss: 1.7446
Test accuracy: 0.3850
```

[3]:
```
from helper import plot_model_evaluation

# Plot model evaluation
plot_model_evaluation(model, x_test, y_test, history, exercise_title='Exercise␣
↪2: CiFAR-10 dataset')
```

```
32/32             0s 1ms/step
```



Training loss Training loss is the calculated error when the model makes predictions on the training data. It is updated after every forward and backward pass of the model during the training process.

- It is expected to decrease during training
- Can give insights into how well the model is learning the training data

Common Training Loss Functions: - Mean Squared Error (MSE): Used for regression tasks. - Cross-Entropy Loss: Common for classification problems.

Validation loss Validation loss evaluates the model's performance on a separate dataset (validation set) that the model has never seen during training. This metric provides an indication of how well the model generalizes to new data.

- Helps in assessing the model's generalization.
- Should decrease initially, but if it starts increasing while training loss decreases, this indicates overfitting.
- Used as criterion for early stopping.

https://www.geeksforgeeks.org/deep-learning/training-and-validation-loss-in-deep-learning/#what-is-training-loss

exercise_3

November 9, 2025

1 Exercise-3: Regularization with Alpha Dropout and MC Dropout

Using the MNIST dataset, extend the previously trained deep neural network by applying Alpha Dropout. Then, without retraining, use Monte Carlo (MC) Dropout at inference to estimate if you can achieve better accuracy. Set random seeds to 42. Use the following configuration:

- Flatten input images to $28 \times 28 = 784$ features
- 3 hidden layers, 64 neurons each
- SELU activation function (required for Alpha Dropout)
- LeCun normal initialization
- Alpha Dropout rate: 0.1 in all hidden layers
- Output layer: 10 neurons with softmax
- Optimizer: Nadam
- learning_rate = 0.001, loss=sparse_categorical_crossentropy
- epochs = 50, batch_size = 32
- Use only the first 1000 training samples and first 200 test samples
- For MC Dropout, enable dropout during inference and average predictions over 20 stochastic forward passes

1.1 Q3.1 Report the test accuracy of the network with Alpha Dropout applied during training.

[9]:
```
from tensorflow.keras.layers import Dense, AlphaDropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import save_model
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.datasets import mnist


import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Set random seeds to 42
tf.random.set_seed(42)
np.random.seed(42)
```

```
# Configuration
loss_function = 'sparse_categorical_crossentropy'
output_activation = 'softmax'
initializer = 'lecun_normal'
learning_rate = 0.001
hidden_neurons = 64
output_neurons = 10
activation = 'selu'
hidden_layers = 3
dropout_rate = 0.1
batch_size = 32
epochs = 50
mc_iterations = 20

# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Use only the first 1000 training samples and first 200 test samples
x_train = x_train[:1000]
y_train = y_train[:1000]
x_test = x_test[:200]
y_test = y_test[:200]

# Preprocess the data - flatten and normalize
x_train = x_train.reshape(-1, 28 * 28).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28 * 28).astype('float32') / 255.0

# Build the model
model = Sequential()

# First hidden layer with input shape and alpha Dropout
model.add(Dense(hidden_neurons, activation=activation,␣
 ↪kernel_initializer=initializer,
                input_shape=(28 * 28,)))
model.add(AlphaDropout(dropout_rate))

# Remaining hidden layers with Alpha Dropout
for _ in range(hidden_layers - 1):
    model.add(Dense(hidden_neurons, activation=activation,␣
 ↪kernel_initializer=initializer))
    model.add(AlphaDropout(dropout_rate))

# Output layer
model.add(Dense(output_neurons, activation=output_activation))
# Compile the model
optimizer = Nadam(learning_rate=learning_rate)
```

```python
model.compile(optimizer=optimizer, loss=loss_function, metrics=['accuracy'])

# Early stopping callback
early_stopping = EarlyStopping(monitor='val_loss',
                     patience=5,
                     restore_best_weights=True)

# Train the model
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    callbacks=[early_stopping],
                    validation_split=0.2,
                    verbose=1)

# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f'Test accuracy: {test_accuracy:.4f}')

# Save the model
save_model(model, 'models/exercise_3_model.keras')
```

```
Epoch 1/50
```

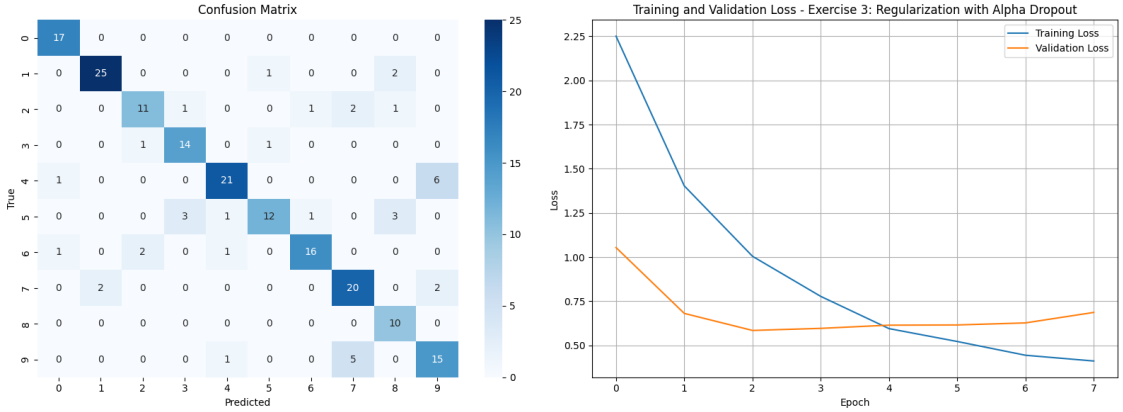
```
/opt/miniconda3/envs/envAssignments/lib/python3.13/site-
packages/keras/src/layers/core/dense.py:92: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
25/25             1s 3ms/step -
accuracy: 0.2438 - loss: 2.1804 - val_accuracy: 0.7300 - val_loss: 1.0059
Epoch 2/50
25/25             0s 1ms/step -
accuracy: 0.5113 - loss: 1.4561 - val_accuracy: 0.8000 - val_loss: 0.6654
Epoch 3/50
25/25             0s 1ms/step -
accuracy: 0.6837 - loss: 0.9770 - val_accuracy: 0.8350 - val_loss: 0.5816
Epoch 4/50
25/25             0s 1ms/step -
accuracy: 0.7350 - loss: 0.7749 - val_accuracy: 0.8500 - val_loss: 0.5689
Epoch 5/50
25/25             0s 1ms/step -
accuracy: 0.7937 - loss: 0.5976 - val_accuracy: 0.8300 - val_loss: 0.5703
Epoch 6/50
25/25             0s 1ms/step -
accuracy: 0.8150 - loss: 0.5687 - val_accuracy: 0.8450 - val_loss: 0.5843
Epoch 7/50
25/25             0s 1ms/step -
```

```
accuracy: 0.8325 - loss: 0.5091 - val_accuracy: 0.8650 - val_loss: 0.5716
Epoch 8/50
25/25             0s 1ms/step -
accuracy: 0.8512 - loss: 0.4162 - val_accuracy: 0.8400 - val_loss: 0.6111
Epoch 9/50
25/25             0s 1ms/step -
accuracy: 0.8750 - loss: 0.3723 - val_accuracy: 0.8500 - val_loss: 0.6572
Test accuracy: 0.8350
```

[8]:
```
from helper import plot_model_evaluation

# Plot model evaluation
plot_model_evaluation(model, x_test, y_test, history, exercise_title='Exercise␣
↪3: Regularization with Alpha Dropout')
```

```
7/7               0s 3ms/step
```



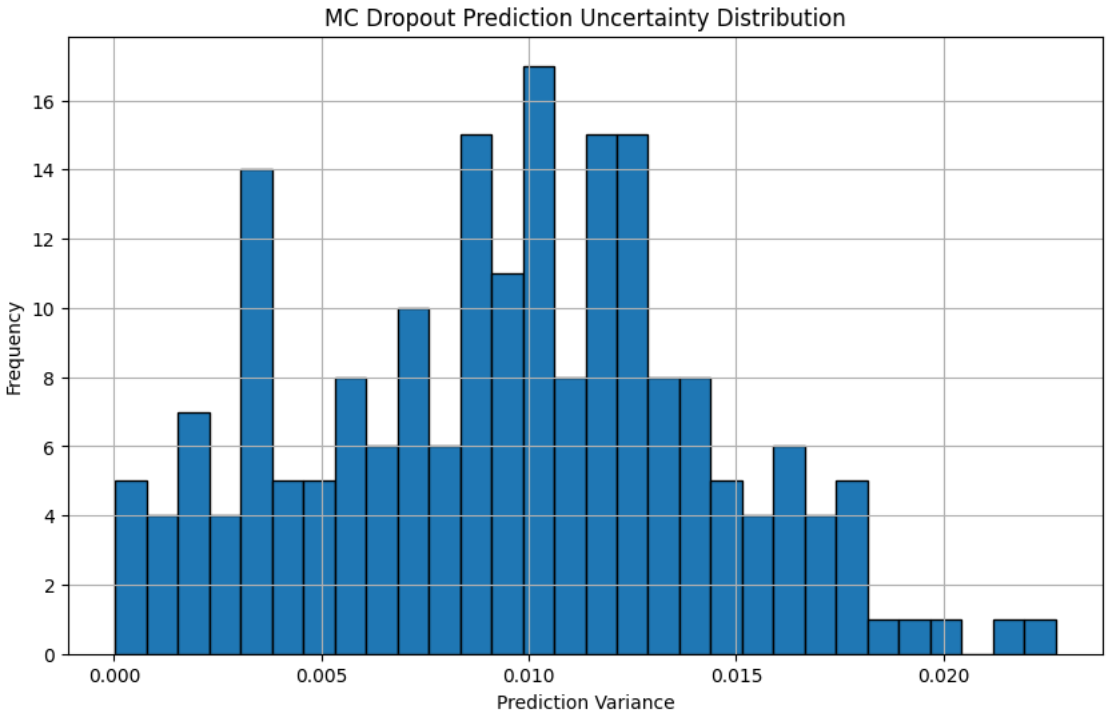
1.2 Q3.2 Report the MC Dropout-enhanced accuracy (averaging 20 stochastic predictions).

[11]:
```
from tensorflow.keras.models import load_model

model = load_model('models/exercise_3_model.keras')

mc_predictions = []
# See configuration above for mc_iterations
for i in range(mc_iterations):
    # Use training=True to enable dropout during inference
    predictions = model(x_test, training=True)
    # Collect predictions
    mc_predictions.append(predictions.numpy())

# Average predictions across all MC iterations
```

```
mc_predictions = np.array(mc_predictions)
mc_mean_predictions = np.mean(mc_predictions, axis=0)

# Get predicted classes from averaged predictions
mc_predicted_classes = np.argmax(mc_mean_predictions, axis=1)

# Calculate MC Dropout accuracy
mc_accuracy = np.mean(mc_predicted_classes == y_test)
print(f'Q3.2 MC Dropout-enhanced accuracy: {mc_accuracy * 100:.2f}%')
```

```
Q3.2 MC Dropout-enhanced accuracy: 84.50%
```

[14]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
prediction_variance = np.var(mc_predictions, axis=0).mean(axis=1)
plt.hist(prediction_variance, bins=30, edgecolor='black')
plt.xlabel('Prediction Variance')
plt.ylabel('Frequency')
plt.title('MC Dropout Prediction Uncertainty Distribution')
plt.grid(True)
plt.show()
```



2 Monte Carlo Dropout

The essence of Monte Carlo (MC) dropout is to embrace uncertainy and randomness, hence the name "Monte Carlo".

During inference (the phase where a trained model is used to make predictions on new, unseen data), instead of turning off dropout (as is standard practice), we keep it active. This means that each time we pass an input through the network, different neurons are randomly dropped out, leading to a variety of outputs for the same input.

This stochastic (randomized) behavior allows us to sample from the model's predictive distribution, and allows us to improve the model's performance by averaging these multiple predictions.

exercise_4

November 9, 2025

1 Exercise-4: Transfer Learning with Pre-trained CNN

Use a pre-trained convolutional neural network (CNN) as a feature extractor and fine-tune a classifier on a subset of the CIFAR-10 dataset. Set random seeds to 42. Follow the configuration below:

- Load CIFAR-10 and normalize pixel values to [0,1]
- Use only the first 2000 training samples and first 500 test samples
- Load MobileNetV2 from tensorflow.keras.applications, with include_top=False and weights='imagenet'
- Freeze all layers of the pre-trained base
- Add a classifier on top:
  - GlobalAveragePooling2D
  - Dense layer with 128 neurons, ReLU activation
  - Dropout: 0.2
  - Output layer: 10 neurons with softmax
- Optimizer: Adam, learning_rate = 0.001
- Loss: sparse_categorical_crossentropy
- epochs = 5, batch_size = 32

1.1 Q4.1 Report the test accuracy of the model.

[2]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.applications import MobileNetV2


# Set random seeds for reproducibility
seed = 42
np.random.seed(seed)
tf.random.set_seed(seed)

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
# Use only the first 2000 training samples and 500 test samples
x_train = x_train[:2000]
y_train = y_train[:2000]
x_test = x_test[:500]
y_test = y_test[:500]

# Resize images from 32x32 to 224x224 for MobileNetV2
x_train = tf.image.resize(x_train, [224, 224])
x_test = tf.image.resize(x_test, [224, 224])

# Load the MobileNetV2 model with pre-trained ImageNet weights
base_model = MobileNetV2(weights='imagenet', include_top=False)

# Freeze the base model
base_model.trainable = False

dropout_rate = 0.2
# Add a classifier on top
inputs = tf.keras.Input(shape=(224, 224, 3))
x = base_model(inputs, training=False)
# Global average pooling layer
x = layers.GlobalAveragePooling2D()(x)
# Dense hidden layer with 128 units and ReLU activation
x = layers.Dense(128, activation='relu')(x)
# 0.2 dropout rate
x = layers.Dropout(0.2)(x)
# Softmax output layer for 10 classes
outputs = layers.Dense(10, activation='softmax')(x)

# Create the final model
model = tf.keras.Model(inputs, outputs)

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Nadam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Configure training parameters
batch_size = 32
epochs = 5

# Train the model
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_split=0.2,
```

```
                    verbose=1)
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f'Test accuracy: {test_accuracy:.4f}')
```

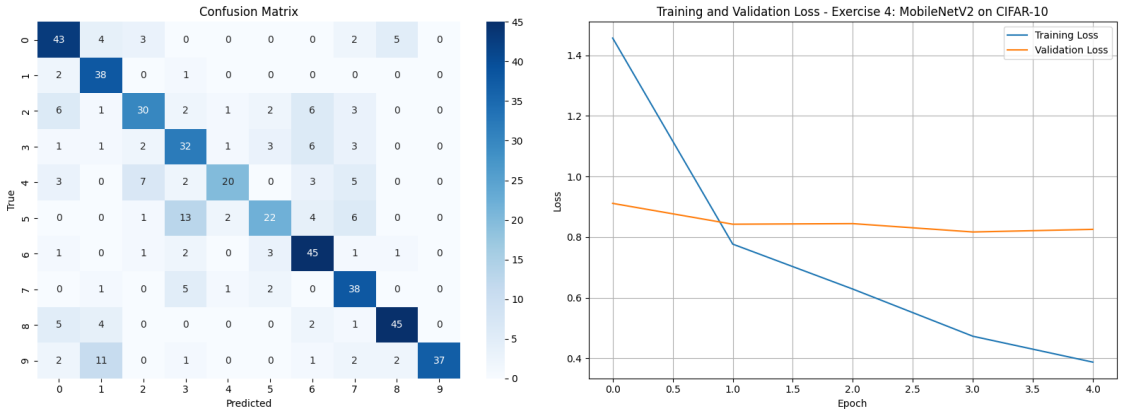
```
/var/folders/_0/wyv54m_x00gfpfh5p7_nchbw0000gn/T/ipykernel_70165/859600164.py:29
: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in
[96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as
the default.
  base_model = MobileNetV2(weights='imagenet', include_top=False)
```

```
Epoch 1/5
50/50             13s 208ms/step -
accuracy: 0.5125 - loss: 1.4571 - val_accuracy: 0.6800 - val_loss: 0.9115
Epoch 2/5
50/50             10s 201ms/step -
accuracy: 0.7300 - loss: 0.7770 - val_accuracy: 0.7050 - val_loss: 0.8427
Epoch 3/5
50/50             10s 199ms/step -
accuracy: 0.7700 - loss: 0.6287 - val_accuracy: 0.7050 - val_loss: 0.8447
Epoch 4/5
50/50             12s 236ms/step -
accuracy: 0.8388 - loss: 0.4730 - val_accuracy: 0.7050 - val_loss: 0.8171
Epoch 5/5
50/50             10s 205ms/step -
accuracy: 0.8744 - loss: 0.3875 - val_accuracy: 0.7125 - val_loss: 0.8257
Test accuracy: 0.7000
```

[4]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np

# Generate predictions and confusion matrix
y_pred = np.argmax(model.predict(x_test), axis=-1)
cm = confusion_matrix(y_test, y_pred)

# Create a figure with 1 row and 2 columns
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# --- Plot 1: Confusion Matrix ---
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[0])
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')
axes[0].set_title('Confusion Matrix')

# --- Plot 2: Training and Validation Loss ---
axes[1].plot(history.history['loss'], label='Training Loss')
```

```
axes[1].plot(history.history['val_loss'], label='Validation Loss')
axes[1].set_xlabel('Epoch')
axes[1].set_ylabel('Loss')
axes[1].set_title('Training and Validation Loss - Exercise 4: MobileNetV2 on␣
 ↪CIFAR-10')
axes[1].legend()
axes[1].grid(True)

# Adjust spacing
plt.tight_layout()
plt.show()
```

```
16/16                 2s 152ms/step
```




```
[ ]:
```

exercise_5

November 9, 2025

1 Exercise-5: Deeper CNN Training on SVHN

Train a controlled deep convolutional neural network (CNN) on a subset of the SVHN dataset. Set random seeds to 42. Load and preprocess SVHN. Build the network using the following configuration:

Data Preparation - Load SVHN and normalize pixel values to [0, 1] - Use only the first 2000 training samples and first 500 test samples - Input shape: 32 × 32 × 3

CNN Architecture - Conv2D: 32 filters, 3×3 kernel, ReLU activation - Conv2D: 32 filters, 3×3 kernel, ReLU activation - MaxPooling2D: 2×2 - Conv2D: 64 filters, 3×3 kernel, ReLU activation - Conv2D: 64 filters, 3×3 kernel, ReLU activation - MaxPooling2D: 2×2 - Flatten - Dense: 256 neurons, ReLU activation - Dropout: 0.3 - Output layer: 10 neurons with softmax activation

Training Configuration - Optimizer: Adam, with learning rate = 0.001 - Loss: sparse_categorical_crossentropy - Epochs: 15 - Batch size: 32

1.1 Data Preparation

1.1.1 The Street View House Numbers (SVHN) Dataset

- Contains over 600,000 labeled digits from real-world images
- 10 classes (digits 0-9), 0 has label 10 from tensorflow.keras.datasets import svhn

[29]:
```
# load SVHN dataset
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Construct a tf.data.Dataset, load only the first 2000 training and 500 test␣
 ↪examples
ds_train, info = tfds.load('svhn_cropped', split='train[:2000]',␣
 ↪as_supervised=True, with_info=True)
# Check the number of examples in the training dataset
print("Number of training examples:", ds_train.cardinality().numpy())
print(info)
```

```
ds_test = tfds.load('svhn_cropped', split='test[:500]', as_supervised=True)
print("Number of test examples:", ds_test.cardinality().numpy())
# print the class names
print("Class names:", info.features['label'].names)

# Display some examples from the dataset
tfds.show_examples(ds_train, info, rows=3, cols=3);

def preprocess(image, label):
    image = tf.cast(image, tf.float32) / 255.0  # Normalize to [0, 1]
    return image, label
ds_train = ds_train.map(preprocess).batch(32).prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(preprocess).batch(32).prefetch(tf.data.AUTOTUNE)

# Build the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(32, 32, 3)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(10, activation='softmax')]
)

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
 ↪patience=5, restore_best_weights=True)

# Train the model
history = model.fit(ds_train, epochs=15,batch_size=32, validation_data=ds_test,␣
 ↪callbacks=[early_stopping])

# Evaluate the model
test_loss, test_accuracy = model.evaluate(ds_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
Number of training examples: 2000
```

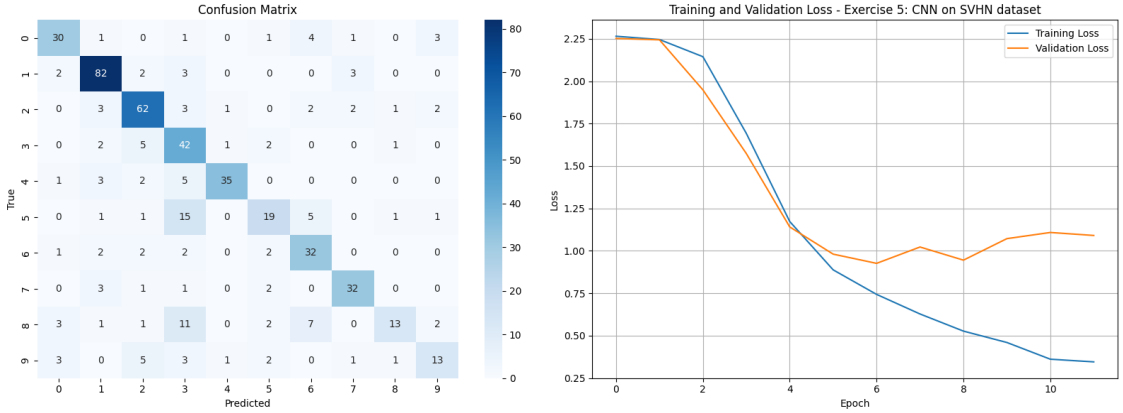
```
tfds.core.DatasetInfo(
    name='svhn_cropped',
    full_name='svhn_cropped/3.1.0',
    description="""
    The Street View House Numbers (SVHN) Dataset is an image digit recognition
    dataset of over 600,000 digit images coming from real world data. Images are
    cropped to 32x32.
    """,
    homepage='http://ufldl.stanford.edu/housenumbers/',
    data_dir='/Users/arthurborgerthorkildsen/tensorflow_datasets/svhn_cropped/3.
1.0',
    file_format=tfrecord,
    download_size=1.47 GiB,
    dataset_size=1.09 GiB,
    features=FeaturesDict({
        'id': Text(shape=(), dtype=string),
        'image': Image(shape=(32, 32, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    nondeterministic_order=False,
    splits={
        'extra': <SplitInfo num_examples=531131, num_shards=8>,
        'test': <SplitInfo num_examples=26032, num_shards=1>,
        'train': <SplitInfo num_examples=73257, num_shards=1>,
    },
    citation="""""Street View House Numbers (SVHN) Dataset, cropped version."""

    @article{Netzer2011,
    author = {Netzer, Yuval and Wang, Tao and Coates, Adam and Bissacco,
Alessandro and Wu, Bo and Ng, Andrew Y},
    booktitle = {Advances in Neural Information Processing Systems ({NIPS})},
    title = {Reading Digits in Natural Images with Unsupervised Feature
Learning},
    year = {2011}
    }""",
)
Number of test examples: 500
Class names: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```


```
Epoch 1/15
63/63             2s 18ms/step -
accuracy: 0.1865 - loss: 2.2650 - val_accuracy: 0.1840 - val_loss: 2.2519
Epoch 2/15
63/63             1s 16ms/step -
accuracy: 0.1955 - loss: 2.2457 - val_accuracy: 0.1840 - val_loss: 2.2439
Epoch 3/15
63/63             1s 17ms/step -
accuracy: 0.2375 - loss: 2.1446 - val_accuracy: 0.3200 - val_loss: 1.9490
Epoch 4/15
63/63             1s 17ms/step -
accuracy: 0.4325 - loss: 1.6924 - val_accuracy: 0.4860 - val_loss: 1.5737
Epoch 5/15
```

```
63/63             1s 17ms/step -
accuracy: 0.6280 - loss: 1.1726 - val_accuracy: 0.6640 - val_loss: 1.1407
Epoch 6/15
63/63             1s 17ms/step -
accuracy: 0.7160 - loss: 0.8877 - val_accuracy: 0.7140 - val_loss: 0.9804
Epoch 7/15
63/63             1s 17ms/step -
accuracy: 0.7625 - loss: 0.7431 - val_accuracy: 0.7200 - val_loss: 0.9261
Epoch 8/15
63/63             1s 17ms/step -
accuracy: 0.7905 - loss: 0.6275 - val_accuracy: 0.7040 - val_loss: 1.0225
Epoch 9/15
63/63             1s 17ms/step -
accuracy: 0.8395 - loss: 0.5266 - val_accuracy: 0.7380 - val_loss: 0.9450
Epoch 10/15
63/63             1s 17ms/step -
accuracy: 0.8400 - loss: 0.4595 - val_accuracy: 0.7440 - val_loss: 1.0717
Epoch 11/15
63/63             1s 17ms/step -
accuracy: 0.8855 - loss: 0.3610 - val_accuracy: 0.7420 - val_loss: 1.1081
Epoch 12/15
63/63             1s 17ms/step -
accuracy: 0.8860 - loss: 0.3451 - val_accuracy: 0.7440 - val_loss: 1.0906
16/16             0s 6ms/step -
accuracy: 0.7200 - loss: 0.9261
Test accuracy: 0.7200
```

[33]:
```python
from helper import plot_model_evaluation
import numpy as np

# Extract x_test and y_test as numpy arrays from the dataset
x_test_list = []
y_test_list = []
for images, labels in ds_test:
    x_test_list.append(images.numpy())
    y_test_list.append(labels.numpy())

x_test = np.concatenate(x_test_list, axis=0)
y_test = np.concatenate(y_test_list, axis=0)

plot_model_evaluation(model, x_test, y_test, history, exercise_title='Exercise␣
↪5: CNN on SVHN dataset')
```

```
16/16             0s 4ms/step
```


Confusion Matrix
True
Predicted
Training and Validation Loss - Exercise 5: CNN on SVHN dataset
Training Loss
Validation Loss
Loss
Epoch

[ ]:

exercise_6

November 9, 2025

1 Exercise 6: CNN with SGD, MC Dropout, and Epistemic Uncertainty

Train a controlled convolutional neural network (CNN) on a subset of the SVHN dataset using the SGD optimizer. Then, apply Monte Carlo (MC) Dropout at inference to estimate both test accuracy and epistemic uncertainty. Set random seeds to 42. Use the following configuration:

Data Preparation - Load SVHN and normalize pixel values to [0, 1] - Use only the first 2000 training samples and first 500 test samples - Input shape: $32 \times 32 \times 3$

CNN Architecture - Conv2D: 32 filters, 3×3 kernel, ReLU activation - MaxPooling2D: 2×2 - Conv2D: 64 filters, 3×3 kernel, ReLU activation - MaxPooling2D: 2×2 - Flatten - Dense: 128 neurons, ReLU activation - Dropout: 0.25 (keep during inference for MC Dropout) - Output layer: 10 neurons with softmax activation

Training Configuration - Optimizer: SGD with momentum = 0.9, learning_rate = 0.01 - Loss: sparse_categorical_crossentropy - Epochs: 15 - Batch size: 32

Monte Carlo Dropout - Enable dropout during inference - Average predictions over 20 stochastic forward passes - Compute the epistemic uncertainty as the predictive variance across passes

1.1 Questions

Q6.1

Report the plain test accuracy of the CNN trained with SGD (no MC Dropout).

Q6.2

Report the MC Dropout–enhanced accuracy (averaging 20 stochastic predictions).

Q6.3

Compute the average epistemic uncertainty (mean predictive variance) across all test samples. Report it as a deterministic number rounded to three decimal places.

[18]:
```
# load SVHN dataset
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds

# Set random seeds for reproducibility
```

```
np.random.seed(42)
tf.random.set_seed(42)

train_size = 2000
test_size = 500

# Construct a tf.data.Dataset, load only the first 2000 training and 500 test␣
 ↪examples
ds_train, info = tfds.load('svhn_cropped', split=f'train[:{train_size}]',␣
 ↪as_supervised=True, with_info=True)
# Check the number of examples in the training dataset
print("Number of training examples:", ds_train.cardinality().numpy())
print(info)
ds_test = tfds.load('svhn_cropped', split=f'test[:{test_size}]',␣
 ↪as_supervised=True)
print("Number of test examples:", ds_test.cardinality().numpy())
# print the class names
print("Class names:", info.features['label'].names)

# Display some examples from the dataset
tfds.show_examples(ds_train, info, rows=3, cols=3);

def preprocess(image, label):
    image = tf.cast(image, tf.float32) / 255.0  # Normalize to [0, 1]
    return image, label
ds_train = ds_train.map(preprocess).batch(32).prefetch(tf.data.AUTOTUNE)
ds_test = ds_test.map(preprocess).batch(32).prefetch(tf.data.AUTOTUNE)

# Build the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(32, 32, 3)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(10, activation='softmax')]
)

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.
 ↪9),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(ds_train, batch_size=32, epochs=15, validation_data=ds_test)
```

```python
# Q6.1
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(ds_test)
print(f"Test Accuracy (SGD): {test_accuracy:.4f}")
```

```
Number of training examples: 2000
tfds.core.DatasetInfo(
    name='svhn_cropped',
    full_name='svhn_cropped/3.1.0',
    description="""
    The Street View House Numbers (SVHN) Dataset is an image digit recognition
    dataset of over 600,000 digit images coming from real world data. Images are
    cropped to 32x32.
    """,
    homepage='http://ufldl.stanford.edu/housenumbers/',
    data_dir='/Users/arthurborgerthorkildsen/tensorflow_datasets/svhn_cropped/3.
1.0',
    file_format=tfrecord,
    download_size=1.47 GiB,
    dataset_size=1.09 GiB,
    features=FeaturesDict({
        'id': Text(shape=(), dtype=string),
        'image': Image(shape=(32, 32, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    nondeterministic_order=False,
    splits={
        'extra': <SplitInfo num_examples=531131, num_shards=8>,
        'test': <SplitInfo num_examples=26032, num_shards=1>,
        'train': <SplitInfo num_examples=73257, num_shards=1>,
    },
    citation="""""Street View House Numbers (SVHN) Dataset, cropped version."""

    @article{Netzer2011,
    author = {Netzer, Yuval and Wang, Tao and Coates, Adam and Bissacco,
Alessandro and Wu, Bo and Ng, Andrew Y},
    booktitle = {Advances in Neural Information Processing Systems ({NIPS})},
    title = {Reading Digits in Natural Images with Unsupervised Feature
Learning},
    year = {2011}
    }""",
)
Number of test examples: 500
Class names: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

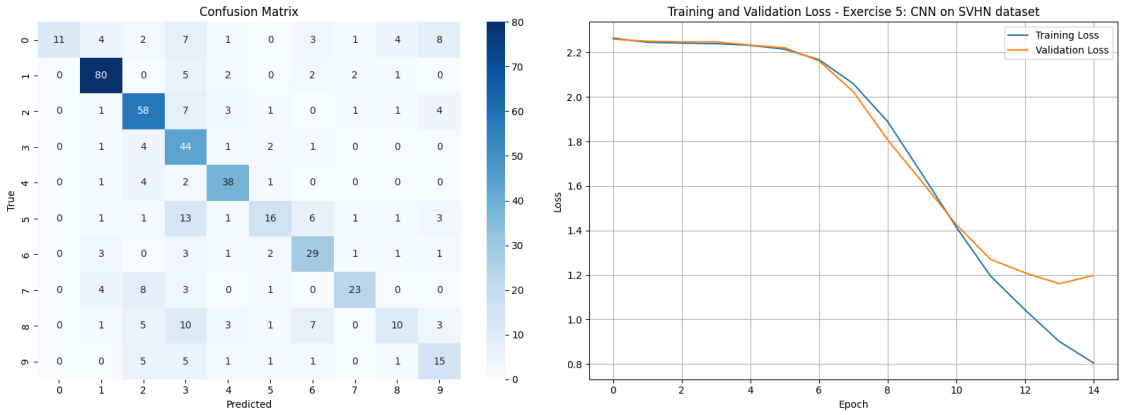

```
Epoch 1/15
63/63             1s 8ms/step -
accuracy: 0.1950 - loss: 2.2639 - val_accuracy: 0.1840 - val_loss: 2.2592
Epoch 2/15
63/63             0s 7ms/step -
accuracy: 0.1955 - loss: 2.2452 - val_accuracy: 0.1840 - val_loss: 2.2500
Epoch 3/15
63/63             0s 7ms/step -
accuracy: 0.1955 - loss: 2.2416 - val_accuracy: 0.1840 - val_loss: 2.2470
Epoch 4/15
63/63             0s 7ms/step -
accuracy: 0.1955 - loss: 2.2393 - val_accuracy: 0.1840 - val_loss: 2.2473
Epoch 5/15
```

```
63/63             0s 7ms/step -
accuracy: 0.1970 - loss: 2.2316 - val_accuracy: 0.1840 - val_loss: 2.2324
Epoch 6/15
63/63             0s 7ms/step -
accuracy: 0.1960 - loss: 2.2140 - val_accuracy: 0.1840 - val_loss: 2.2207
Epoch 7/15
63/63             0s 7ms/step -
accuracy: 0.2160 - loss: 2.1672 - val_accuracy: 0.2280 - val_loss: 2.1621
Epoch 8/15
63/63             0s 6ms/step -
accuracy: 0.2775 - loss: 2.0592 - val_accuracy: 0.3140 - val_loss: 2.0232
Epoch 9/15
63/63             0s 6ms/step -
accuracy: 0.3575 - loss: 1.8888 - val_accuracy: 0.4120 - val_loss: 1.8070
Epoch 10/15
63/63             0s 7ms/step -
accuracy: 0.4520 - loss: 1.6538 - val_accuracy: 0.4680 - val_loss: 1.6184
Epoch 11/15
63/63             0s 6ms/step -
accuracy: 0.5410 - loss: 1.4149 - val_accuracy: 0.5480 - val_loss: 1.4264
Epoch 12/15
63/63             0s 7ms/step -
accuracy: 0.6155 - loss: 1.1949 - val_accuracy: 0.5980 - val_loss: 1.2702
Epoch 13/15
63/63             0s 7ms/step -
accuracy: 0.6580 - loss: 1.0435 - val_accuracy: 0.6320 - val_loss: 1.2098
Epoch 14/15
63/63             0s 7ms/step -
accuracy: 0.7050 - loss: 0.9015 - val_accuracy: 0.6480 - val_loss: 1.1612
Epoch 15/15
63/63             0s 7ms/step -
accuracy: 0.7330 - loss: 0.8047 - val_accuracy: 0.6480 - val_loss: 1.1980
16/16             0s 3ms/step -
accuracy: 0.6480 - loss: 1.1980
Test Accuracy (SGD): 0.6480
```

[21]:
```python
from helper import plot_model_evaluation
import numpy as np

# Extract x_test and y_test as numpy arrays from the dataset
x_test_list = []
y_test_list = []
for images, labels in ds_test:
    x_test_list.append(images.numpy())
    y_test_list.append(labels.numpy())

x_test = np.concatenate(x_test_list, axis=0)
```

```
y_test = np.concatenate(y_test_list, axis=0)

plot_model_evaluation(model, x_test, y_test, history, exercise_title='Exercise␣
↪5: CNN on SVHN dataset')
```

```
16/16                0s 3ms/step
```



1.2 Q6.2

Report the MC Dropout–enhanced accuracy (averaging 20 stochastic predictions).

[19]:
```
# Q6.2: MC Dropout - Enable dropout during inference
import numpy as np

# Extract test data as numpy arrays
x_test_list = []
y_test_list = []
for images, labels in ds_test:
    # Append batch data to lists
    x_test_list.append(images.numpy())
    y_test_list.append(labels.numpy())

# Concatenate all batches to form complete test sets
x_test = np.concatenate(x_test_list, axis=0)
y_test = np.concatenate(y_test_list, axis=0)

# Perform 20 stochastic forward passes with dropout enabled
mc_iterations = 20
mc_predictions = []

for i in range(mc_iterations):
    # Use training=True to enable dropout during inference (make predictions on␣
↪unseen data)
```

```
    predictions = model(x_test, training=True)
    mc_predictions.append(predictions.numpy())

# Average predictions across all MC iterations
mc_predictions = np.array(mc_predictions)
mc_mean_predictions = np.mean(mc_predictions, axis=0)

# Get predicted classes from averaged predictions
mc_predicted_classes = np.argmax(mc_mean_predictions, axis=1)

# Calculate MC Dropout accuracy, compare with true labels
mc_accuracy = np.mean(mc_predicted_classes == y_test)
# Print 5 first predicted classes and true labels for verification
print("First 5 MC Dropout predicted classes:", mc_predicted_classes[:5])
print("First 5 true labels:", y_test[:5])
print(f'Q6.2 MC Dropout-enhanced accuracy: {mc_accuracy * 100:.2f}%')
```

```
First 5 MC Dropout predicted classes: [2 7 2 4 4]
First 5 true labels: [2 7 2 4 4]
Q6.2 MC Dropout-enhanced accuracy: 63.60%
```

The enhanced MC Dropout accuracy means that we take the average of the predictions from multiple stochastic forward passes through the network with dropout enabled during inference. This helps to capture the uncertainty in the model's predictions and can lead to improved accuracy.

1.3 Q6.3

Compute the average epistemic uncertainty (mean predictive variance) across all test samples. Report it as a deterministic number rounded to three decimal places.

[20]:
```
prediction_variance = np.var(mc_predictions, axis=0)

# Mean variance across all classes for each sample
mean_variance_per_sample = np.mean(prediction_variance, axis=1)

# Average epistemic uncertainty across all test samples
epistemic_uncertainty = np.mean(mean_variance_per_sample)
print(f'Q6.3 Average epistemic uncertainty: {epistemic_uncertainty:.3f}')
```

```
Q6.3 Average epistemic uncertainty: 0.006
```

The average epistemic uncertainty is calculated by measuring the variance in the predictions across the multiple stochastic forward passes. This variance reflects the model's uncertainty about its predictions, and averaging it across all test samples provides a single measure of uncertainty for the entire test set.