1. Sequential model for MNIST

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')])

model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])
```

2. CNN model for grayscale images

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),
    Conv2D(64, (3,3), activation='relu'),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')])
```

3. Add Dense(64) before output

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

new_model = Sequential(model.layers[:-1])
new_model.add(Dense(64, activation='relu'))
new_model.add(model.layers[-1])

new_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

4. Sigmoid vs Softmax + 3-class example

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Sigmoid: independent probabilities (multi-label)
# Softmax: probabilities sum to 1 (multi-class)

model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),
    Dense(3, activation='softmax')])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

5. Functional API with two inputs

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Concatenate

input1 = Input(shape=(32,))
input2 = Input(shape=(32,))
merged = Concatenate()([input1, input2])
x = Dense(64, activation='relu')(merged)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=[input1, input2], outputs=output)
```

6. Load CIFAR-10 dataset

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

7. Reshape dataset for CNN input

```
import numpy as np

X = np.random.rand(1000, 28, 28)
X_cnn = X.reshape((1000, 28, 28, 1))
```

8. Split dataset 70/15/15

```
from sklearn.model_selection import train_test_split
import numpy as np

X = np.random.rand(1000, 28, 28, 1)
y = np.random.randint(0, 10, 1000)

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)
```

9. Data augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True)
```

10. Visualize 5 random images

```
import matplotlib.pyplot as plt
import numpy as np

X = np.random.rand(20, 28, 28)
y = np.arange(20)

indices = np.random.choice(len(X), 5, replace=False)
for i in indices:
    plt.imshow(X[i], cmap='gray')
    plt.title(f"Label: {y[i]}")
    plt.axis('off')
    plt.show()
```

11. Train model with EarlyStopping

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
import numpy as np

X_train = np.random.rand(100, 28, 28)
y_train = np.random.randint(0, 10, 100)
X_val = np.random.rand(20, 28, 28)
y_val = np.random.randint(0, 10, 20)

y_train_cat = to_categorical(y_train, 10)
y_val_cat = to_categorical(y_val, 10)

model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

es = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
model.fit(X_train, y_train_cat, validation_data=(X_val, y_val_cat), epochs=10,
callbacks=[es])
```

12. ModelCheckpoint to save best model based on validation accuracy

```
from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',
save_best_only=True, mode='max')
model.fit(X_train, y_train_cat, validation_data=(X_val, y_val_cat), epochs=10,
callbacks=[checkpoint])
```

13. Plot training & validation loss curves

```
import matplotlib.pyplot as plt

history = model.history  # assume model has been trained

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

14. Evaluate trained model on test data

```
X_test = np.random.rand(20, 28, 28)
y_test = np.random.randint(0, 10, (20,))
from tensorflow.keras.utils import to_categorical
y_test_cat = to_categorical(y_test, 10)

loss, accuracy = model.evaluate(X_test, y_test_cat)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

15. Detect overfitting from loss arrays

```
train_loss = [0.8, 0.5, 0.3, 0.2]
val_loss   = [0.9, 0.6, 0.4, 0.5]

if val_loss[-1] > val_loss[-2]:
    print("Overfitting detected: Validation loss increased while training loss
decreased.")
else:
    print("No overfitting detected.")
```

16. Freeze all layers except last Dense layer

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Example pre-trained CNN
model = Sequential([
    Dense(64, activation='relu', input_shape=(100,)),
    Dense(10, activation='softmax')])

for layer in model.layers[:-1]:
    layer.trainable = False

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

17. Extract output of intermediate layer using functional API

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
import numpy as np

inp = Input(shape=(10,))
x = Dense(16, activation='relu')(inp)
mid = Dense(8, activation='relu', name='intermediate')(x)
out = Dense(1, activation='sigmoid')(mid)

model = Model(inputs=inp, outputs=out)

# Model to extract intermediate layer output
intermediate_model = Model(inputs=model.input,
outputs=model.get_layer('intermediate').output)

sample = np.random.rand(1, 10)
print(intermediate_model.predict(sample))
```

18. Custom MSE loss function

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def custom_mse(y_true, y_pred):
    return tf.reduce_mean(tf.square(y_true - y_pred))

model = Sequential([
    Dense(16, activation='relu', input_shape=(5,)),
    Dense(1)
])

model.compile(optimizer='adam', loss=custom_mse)
```

19. Save and load model

```
# Save
model.save('model.h5')          # HDF5 format
model.save('saved_model/')      # SavedModel format

# Load
from tensorflow.keras.models import load_model
loaded_h5 = load_model('model.h5')
loaded_saved = load_model('saved_model/')
```

20. Apply softmax manually and with TensorFlow

```
import numpy as np
import tensorflow as tf

logits = np.array([2.0, 1.0, 0.1])

# Manual softmax
exp_vals = np.exp(logits - np.max(logits))
manual_softmax = exp_vals / np.sum(exp_vals)
print("Manual softmax:", manual_softmax)

# TensorFlow softmax
tf_softmax = tf.nn.softmax(logits).numpy()
print("TensorFlow softmax:", tf_softmax)
```