In [4]:

```
pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.12.0-cp39-cp39-win_amd64.whl (1.9 kB)
Collecting tensorflow-intel==2.12.0
  Downloading tensorflow_intel-2.12.0-cp39-cp39-win_amd64.whl (272.8 M
B)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\jos
ep\anaconda3\lib\site-packages (from tensorflow-intel==2.12.0->tensorfl
ow) (3.10.0.2)
Collecting termcolor>=1.1.0
  Downloading termcolor-2.3.0-py3-none-any.whl (6.9 kB)
Collecting jax>=0.3.15
  Downloading jax-0.4.8.tar.gz (1.2 MB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
    Preparing wheel metadata: started
    Preparing wheel metadata: finished with status 'done'
Requirement already satisfied: setuptools in c:\users\josep\anaconda3\l
ib\site-packages (from tensorflow-intel==2.12.0->tensorflow) (58.0.4)
```

In [19]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam

np.random.seed(2)
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, pre
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau


sns.set(style='white', context='notebook', palette='deep')
```

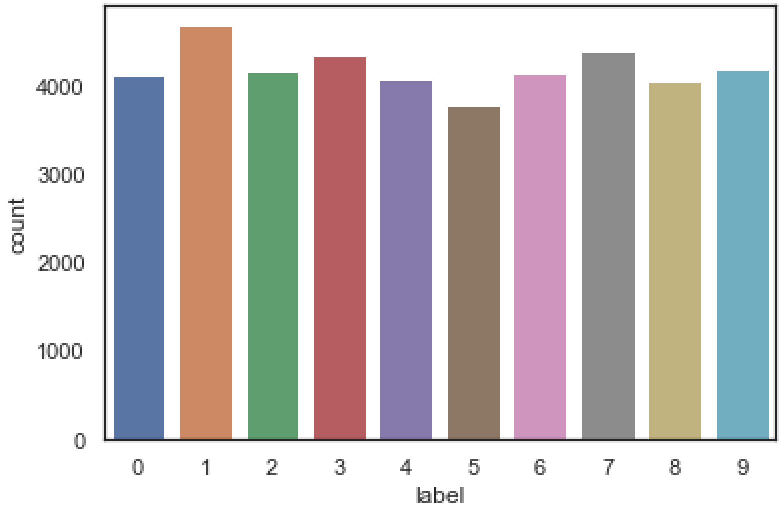
In [2]:

```
# Load the data
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
Y_train = train["label"]

# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)

# free some space
del train

g = sns.countplot(Y_train)

Y_train.value_counts()
```

```
C:\Users\josep\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or misinterp
retation.
  warnings.warn(
```

Out[2]:

```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```



In [3]:

```
# Check the data
X_train.isnull().any().describe()
# Null Values
test.isnull().any().describe()
```

Out[3]:

```
count       784
unique        1
top       False
freq        784
dtype: object
```

In [4]:

```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
```

In [5]:

```
# Reshape image in 3 dimensions (height = 28px, width = 28px , canal = 1)
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
```

In [6]:

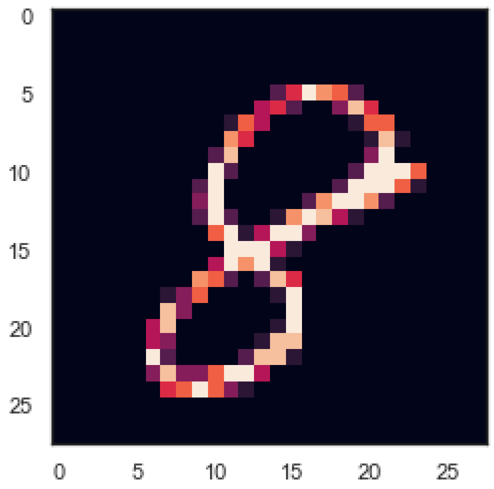
```
# Encode labels to one hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
Y_train = to_categorical(Y_train, num_classes = 10)
```

In [7]:

```
random_seed=2
# Split the train and the validation set for the fitting
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, ran
```

In [8]:

```
# Some examples
g = plt.imshow(X_train[0][:,:,0])
```



In [9]:

```
num_classes = 10

# Define the CNN model
# 5 layers build to identify numbers
def create_model():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))


    return model

# Create an instance of the model
model = create_model()

#Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model on the dataset
model.fit(X_train, Y_train, epochs=10, validation_data=(X_val, Y_val))

# Save the model for use with Edge Impulse
model.save('model.h5')
```

```
Epoch 1/10
1182/1182 [==============================] - 11s 9ms/step - loss: 0.0591 -
accuracy: 0.9008 - val_loss: 0.0210 - val_accuracy: 0.9702
Epoch 2/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0165 -
accuracy: 0.9758 - val_loss: 0.0153 - val_accuracy: 0.9779
Epoch 3/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0118 -
accuracy: 0.9814 - val_loss: 0.0155 - val_accuracy: 0.9731
Epoch 4/10
1182/1182 [==============================] - 10s 8ms/step - loss: 0.0089 -
accuracy: 0.9858 - val_loss: 0.0130 - val_accuracy: 0.9802
Epoch 5/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0073 -
accuracy: 0.9886 - val_loss: 0.0104 - val_accuracy: 0.9824
Epoch 6/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0057 -
accuracy: 0.9909 - val_loss: 0.0111 - val_accuracy: 0.9855
Epoch 7/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0050 -
accuracy: 0.9921 - val_loss: 0.0109 - val_accuracy: 0.9871
Epoch 8/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0039 -
accuracy: 0.9945 - val_loss: 0.0108 - val_accuracy: 0.9862
Epoch 9/10
1182/1182 [==============================] - 9s 8ms/step - loss: 0.0033 -
accuracy: 0.9953 - val_loss: 0.0106 - val_accuracy: 0.9860
Epoch 10/10
1182/1182 [==============================] - 13s 11ms/step - loss: 0.0026
- accuracy: 0.9961 - val_loss: 0.0106 - val_accuracy: 0.9867
```

In [10]:

```
# Define the optimizer
optimizer = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08, decay=0.0)
#Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accur
```

In [12]:

```
#Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                           patience=3,
                                          verbose=1,
                                          min_lr=0.00001)
```

In [13]:

```
# Data augmentation to prevent overfitting
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image 
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total 
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total h
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(X_train)
```

In [14]:

```
epochs = 1
batch_size = 86
```

In [15]:

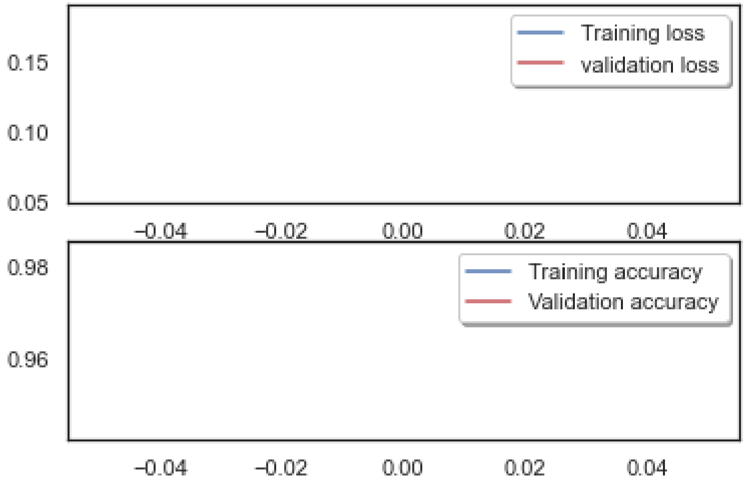
```
# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (X_val,Y_val),
                              verbose = 2, steps_per_epoch=X_train.shape[0] // batch_siz
                              , callbacks=[learning_rate_reduction])
```

```
C:\Users\josep\AppData\Local\Temp/ipykernel_149964/923181679.py:2: UserWar
ning: `Model.fit_generator` is deprecated and will be removed in a future
version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(datagen.flow(X_train,Y_train, batch_size=b
atch_size),
```

```
439/439 - 19s - loss: 0.1853 - accuracy: 0.9446 - val_loss: 0.0563 - val_a
ccuracy: 0.9831 - lr: 0.0010 - 19s/epoch - 44ms/step
```

In [16]:

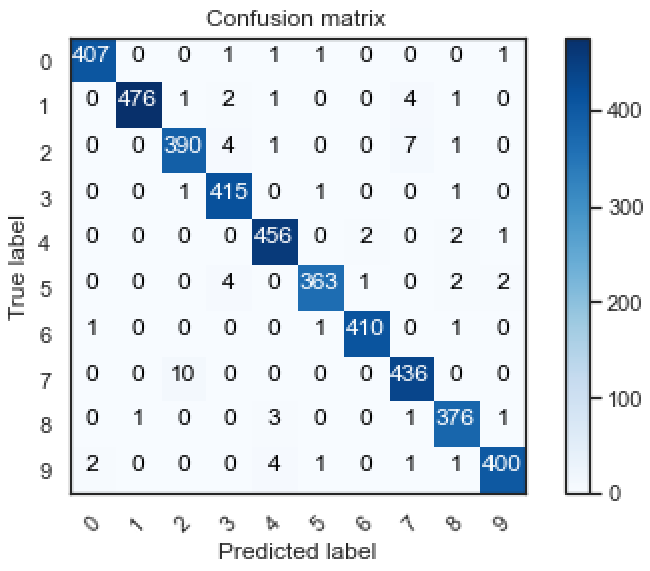
```
# Plot the loss and accuracy curves for training and validation
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



In [17]:

```
# Confusion matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

```
132/132 [==============================] - 1s 5ms/step
```



	0	1	2	3	4	5	6	7	8	9
0	407	0	0	1	1	1	0	0	0	1
1	0	476	1	2	1	0	0	4	1	0
2	0	0	390	4	1	0	0	7	1	0
3	0	0	1	415	0	1	0	0	1	0
4	0	0	0	0	456	0	2	0	2	1
5	0	0	0	4	0	363	1	0	2	2
6	1	0	0	0	0	1	410	0	1	0
7	0	0	10	0	0	0	0	436	0	0
8	0	1	0	0	3	0	0	1	376	1
9	2	0	0	0	4	1	0	1	1	400