



- ▼ Project 2- ECE 188, Spring 2022 - Deadline: June 10th, 2022.
Please upload a link to your Github on Gradescope.

Train a Deep Learning model on the MNIST dataset and evaluate it on a test set.

The first part in this notebook will take you through the training of the [MNIST Dataset](#) which is a collection of handwritten digits.

We will build a Deep Learning based system that can recognise handwritten digits using [Keras](#) and [Tensorflow](#).

Note that the entire project will be done on a Google Colab. A tutorial to some basic colab instructions can be found [here](#).

Note that all the code needed is provided until Task 1, **you will only have to write code after the task 1 header**

▼ Import Necessary Packages

We import Keras and Tensorflow to train our network and numpy to work with our data.

We also import matplotlib to view our images.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
mpl.rcParams['figure.figsize'] = (8, 8)
mpl.rcParams['axes.grid'] = False
```

▼ Load the Data

We perform some data preprocessing. Feel free to skip over this and use the train and test data as `x_train` and `x_test`

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist11493376/11490434 [=====] - 0s 0us/step
11501568/11490434 [=====] - 0s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

▼ Define a model in keras.

We define a simple simple network with 2 hidden layers. One fully connected dense layer with 8 neurons and onr flatten layer that flattens the model into 1D structure which is then fed into a softmax output.

This is a barebones keras structure. In task 1 you will be tasked with improving this model.

```
model = keras.Sequential(
    [
```

```

keras.Input(shape=input_shape),
layers.Dense(8,activation="relu"),
layers.Flatten(),
layers.Dense(num_classes, activation="softmax"),
]
)

```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 28, 28, 8)	16
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 10)	62730
Total params: 62,746		
Trainable params: 62,746		
Non-trainable params: 0		

▼ Compile and train the model

We know train our model. Before we do that we need to set some parameters to train the model. Hence we define the loss and optimizer to be used as well as batch size. The number of epochs refers to the number of training iterations.

```
batch_size = 128
epochs = 15
```

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

```
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```

Epoch 1/15
422/422 [=====] - 5s 4ms/step - loss: 0.5376 - accuracy: 0.866
Epoch 2/15
422/422 [=====] - 2s 4ms/step - loss: 0.3106 - accuracy: 0.911
Epoch 3/15
422/422 [=====] - 2s 4ms/step - loss: 0.2882 - accuracy: 0.918
Epoch 4/15
422/422 [=====] - 1s 4ms/step - loss: 0.2779 - accuracy: 0.921
Epoch 5/15
422/422 [=====] - 2s 4ms/step - loss: 0.2711 - accuracy: 0.923

```

```

Epoch 6/15
422/422 [=====] - 2s 4ms/step - loss: 0.2669 - accuracy: 0.925
Epoch 7/15
422/422 [=====] - 2s 5ms/step - loss: 0.2640 - accuracy: 0.925
Epoch 8/15
422/422 [=====] - 2s 4ms/step - loss: 0.2598 - accuracy: 0.928
Epoch 9/15
422/422 [=====] - 2s 4ms/step - loss: 0.2574 - accuracy: 0.927
Epoch 10/15
422/422 [=====] - 2s 4ms/step - loss: 0.2558 - accuracy: 0.928
Epoch 11/15
422/422 [=====] - 2s 4ms/step - loss: 0.2538 - accuracy: 0.929
Epoch 12/15
422/422 [=====] - 1s 4ms/step - loss: 0.2527 - accuracy: 0.929
Epoch 13/15
422/422 [=====] - 2s 4ms/step - loss: 0.2505 - accuracy: 0.929
Epoch 14/15
422/422 [=====] - 1s 4ms/step - loss: 0.2505 - accuracy: 0.930
Epoch 15/15
422/422 [=====] - 1s 4ms/step - loss: 0.2484 - accuracy: 0.930
<keras.callbacks.History at 0x7fa3000ac510>

```

▼ Evaluate the model

```

score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

```

```

Test loss: 0.27171605825424194
Test accuracy: 0.9257000088691711

```

We achieve an accuracy on the test set of 92% but can we do better?

▼ Looking at a prediction.

Lets look at the prediction for a particular image in our model. We will look at one corect and one incorrect prediction and see if we can improve on the incorrect prediction.

```

def get_mnist_label(image_probs):

    return np.argmax(image_probs), np.max(image_probs)

preds = model.predict(x_test)

```

```

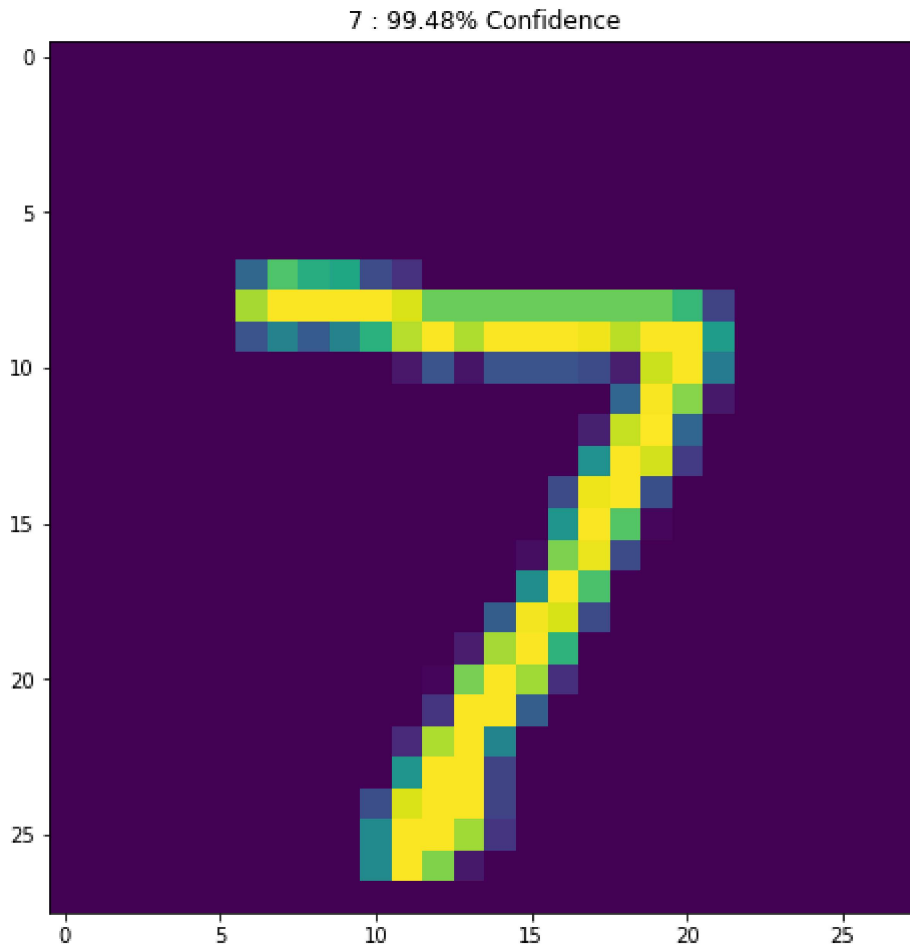
image = x_test[0].reshape(input_shape[0], input_shape[1])
label = np.argmax(y_test[0])
image_probs = preds[0]

```

```

plt.figure()
plt.imshow(image*0.5 + 0.5) # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(image_probs)
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()

```



```

# Store the incorrect prediction index here
incorrect_index = 0

```

```

for i in range(preds.shape[0]):

```

```

    if np.argmax(preds[i]) != np.argmax(y_test[i]):

```

```

        incorrect_index = i

```

```

image_incorrect = x_test[incorrect_index].reshape(input_shape[0], input_shape[1])

```

```

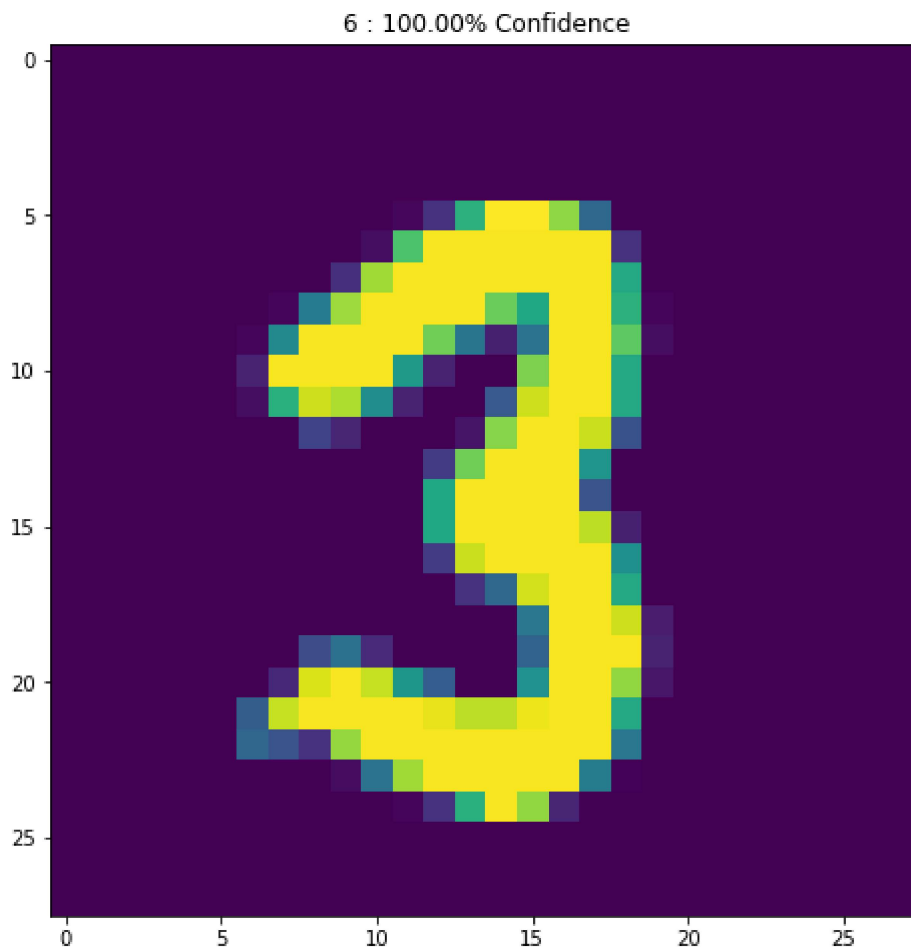
plt.figure()

```

```

plt.imshow(image_incorrect*0.5 + 0.5) # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(preds[i])
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()

```



We see that a 3 has been predicted with high confidence as a 6 does this change with our improved model below?

▼ TASK 1: Improve the accuracy of the model

Your goal in task one is to improve the accuracy of the model on the test set.

For image recognition tasks convolutional neural networks work better than most other type of architectures. Your goal is to push the test accuracy above 99% percent.

The following steps will get you there.

1. Convert the 1st hidden Dense layer to a conv layer with 32 filters, a filter size of 3x3 and with relu activation. (I've done this step)

2. Add a Maxpooling layer of size 2x2.
3. Add another Conv layer with 64 filters, with a filter size of 3x3 with relu activation.
4. Add a Maxpooling layer of size 2x2.
5. Add Dropout to the layer after the Flatten layer.

```
improved_model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        # Add Maxpooling layer here
        layers.MaxPooling2D(strides=(2,2), padding='same'),
        # Add Conv2d layer here
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.Flatten(),
        # Add dropout of 0.5 here
        layers.Dropout(0.2, input_shape=(2,)),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

improved_model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_5 (Conv2D)	(None, 11, 11, 64)	18496
flatten_3 (Flatten)	(None, 7744)	0
dropout (Dropout)	(None, 7744)	0
dense_4 (Dense)	(None, 10)	77450
Total params: 96,266		
Trainable params: 96,266		
Non-trainable params: 0		

▼ Compile and train the improved model.

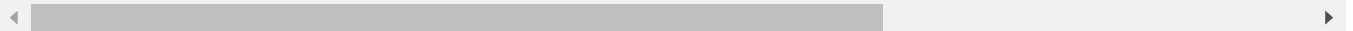
batch size = 128

```
epochs = 15
```

```
improved_model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
```

```
improved_model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/15
422/422 [=====] - 12s 6ms/step - loss: 0.2436 - accuracy: 0.92
Epoch 2/15
422/422 [=====] - 2s 5ms/step - loss: 0.0690 - accuracy: 0.979
Epoch 3/15
422/422 [=====] - 2s 5ms/step - loss: 0.0488 - accuracy: 0.985
Epoch 4/15
422/422 [=====] - 2s 5ms/step - loss: 0.0388 - accuracy: 0.987
Epoch 5/15
422/422 [=====] - 2s 5ms/step - loss: 0.0321 - accuracy: 0.990
Epoch 6/15
422/422 [=====] - 2s 5ms/step - loss: 0.0266 - accuracy: 0.991
Epoch 7/15
422/422 [=====] - 2s 5ms/step - loss: 0.0237 - accuracy: 0.992
Epoch 8/15
422/422 [=====] - 2s 5ms/step - loss: 0.0199 - accuracy: 0.993
Epoch 9/15
422/422 [=====] - 2s 5ms/step - loss: 0.0169 - accuracy: 0.994
Epoch 10/15
422/422 [=====] - 2s 5ms/step - loss: 0.0141 - accuracy: 0.995
Epoch 11/15
422/422 [=====] - 2s 5ms/step - loss: 0.0133 - accuracy: 0.995
Epoch 12/15
422/422 [=====] - 2s 5ms/step - loss: 0.0109 - accuracy: 0.996
Epoch 13/15
422/422 [=====] - 2s 5ms/step - loss: 0.0096 - accuracy: 0.996
Epoch 14/15
422/422 [=====] - 2s 5ms/step - loss: 0.0091 - accuracy: 0.997
Epoch 15/15
422/422 [=====] - 2s 5ms/step - loss: 0.0076 - accuracy: 0.997
<keras.callbacks.History at 0x7fa27a35a310>
```



▼ Evaluate the Improved model.

You should achieve a test accuracy of greater than 99%!

If not the code will throw an assertion error.

```
score = improved_model.evaluate(x_test, y_test, verbose=0)
print("Improved Test loss:", score[0])
print("Improved Test accuracy:", score[1])
```

```
assert score[1] > 0.99
```


Improved Test loss: 0.03831860423088074
Improved Test accuracy: 0.9905999898910522

▼ Looking at a prediction.

Lets look at the prediction for a particular image in our model. We will look at one corect and one incorrect prediction and see if we can improve on the incorrect prediction.

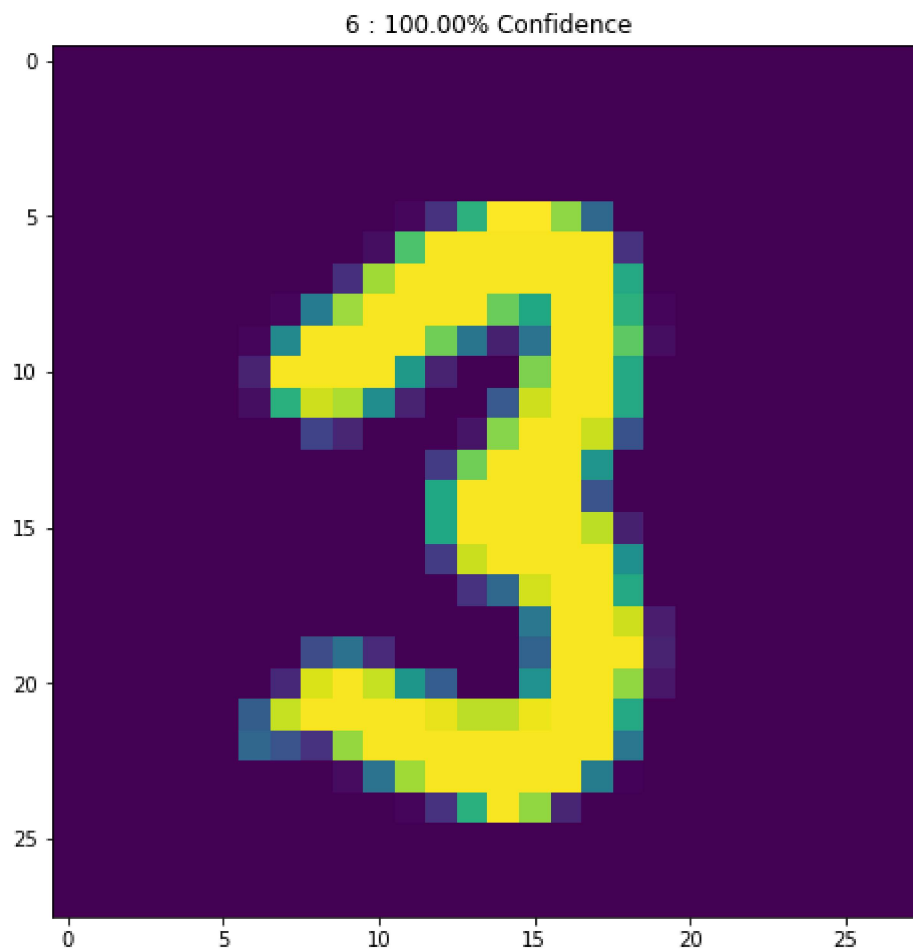
Check the predictions on the improved model, do you see improvements on the incorrect sample.

```
improved_preds = improved_model.predict(x_test)

image = x_test[0].reshape(input_shape[0], input_shape[1])
label = np.argmax(y_test[0])

plt.figure()
plt.imshow(image*0.5 + 0.5) # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(improved_preds[0])
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```

```
plt.figure()
plt.imshow(image_incorrect*0.5 + 0.5) # To change [-1, 1] to [0,1]
image_class, class_confidence = get_mnist_label(improved_preds[i])
plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
plt.show()
```



✓ 0s completed at 10:07 AM

● ✕