

Note: You will be scored (given points) on this exercise per the syllabus, so make sure to attempt all exercise parts.

Exercise 1: Object Oriented Programming (OOP)

Reading and preparation

- a. The purpose of this preparation is to get you familiar with the basics of what object oriented programming is, and what its key concepts are.
- b. Read through the subset of the lecture slides for OOP in python
 - i. See: `lecture_materials/Montillo_Talk_OOP_TensorflowKeras_vx.x.pdf`
- c. Review OOP in python by studying the examples in
 - i. See: `tasks/OOP/OOP_in_Python.ipynb`

Exercise 2: Manipulating images with MyImageClass

Reading and preparation:

- The purpose of this preparation is to get you familiar with the basics of what an image is including basic properties like image dimensions and resolution, and basic operations one might apply to an image such as loading and plotting (displaying) them, and computing operations on images like computing the difference of two images
- Read through the lecture slides for ImageIO in python
 - ii. See: `lecture_materials/ImageIO_slides_vx.x.pdf`

Exercise:

- `MyImageClass.py`
 - All of the above concepts are illustrated with an object oriented programmed class called `MyImageClass`
 - Contains the custom class for the exercise and used in the above notebook
 - `MyImageClass.ipynb`
 - The code to step through various capabilities in our class, and look at figures
 - There are 5 main parts of the exercise.
 - Parts 1 and 2 just go over the examples used in lecture
 - Provide an example of how to use Matplotlib with multiple axis
 - Part 3 focuses image analysis on MNIST (hand written digits)
 - Part 4 focuses on analysis to compare slices of a 3D MRI volumetric image
 - Part 5 compares reconstructed images from the original using MSE (mean squared error)
 - Parts 3,4,5 use a custom class “`MyImgClass`” that contains useful functions for the analysis.
 - **You will need to fill in the missing functions/methods for this custom class.**
- a. Do some basic image analysis on MNIST (the hand written digit dataset)
- 1)Load data
 - 2)Create objects of `MyImgClass` for each number
 - 3)visualize an example of each digit using the `MyImgClass.fPlot` method
 - 4)Create mean and std images and plot
 - 5)Create mean and std images per digit label and plot
 - 6)Look at example difference images and MSE
 - i. Difference image using `MyImgClass.fPixelwiseSqDif`
 - Plot difference image
 - Calculate and output MSE using `MyImgClass.fMSE`

ii.

Difference images for all pairs of mean digits
Calculate and plot the MSE for each pair of mean digits

- b. Image Analysis on coronal MRI slice of normal subjects, vs subjects with Alzheimer's disease (AD)
- 1) load data
 - 2) create MyImgClass for each image, plot an example
 - 3) Plot and example normal brain (label 0) and AD brain (label 1)
 - 4) Plot mean images for both normal and AD brains
 - 5) Calculate and plot the difference of the mean images.
 - 6) Calculate pairwise MSE between the normal and AD brains
- c. Now compare images before and after they have been reconstructed by an autoencoder
- 1) For each of the 100 epochs load the images, calculate MSE, and plot to compare the original and recon images.
 - i. For every tenth epoch, plot the images to compare
 - 2) Plot the MSE per epoch as a line plot
 - 3) For the last epoch, calculate the MSE between the original and reconstructed images, and compare our MSE calculation to the keras implementation
- **Note: the keras loss function takes the mean MSE across a batch, so we'll need to take the mean MSE.****

Exercise 3: Tensorflow and Keras libraries for building basic neural networks

Reading and preparation:

- a. Read through the subset of the lecture slides for Tensorflow/Keras
 - i. See: [lecture_materials/Montillo_Talk_OOP_TensorflowKeras_vx.x.pdf](#)

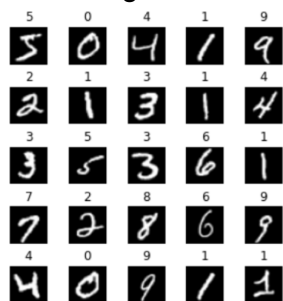
Part 0: Get familiar with tensors (the N-dimensional extension to vectors in matrices) that are used throughout NNs by studying the examples in “tensorBasics.ipynb”

Part 1: Open and explore the code in “neuralnetBasics.ipynb” then extend it as described below.

Part 1 of this exercise uses the MNIST dataset.

MNIST has 60,000 training images and 10,000 test images. Each image is 28x28 pixel grayscale (single channel) image.

Each image is one of 10 handwritten digits.



The challenge is to build a classifier that reads the image and predicts which digit it contains.















































































































The file contains a sequential model to perform that classification.

- b. **ToDo #1.** Your first task is to study the code and then list what objects it is built from. List them in your code where the comment says **ToDo #1**.
- c. **ToDo #2.** Now fit the model and write down what train and test accuracy you obtain.

- d. ToDo #3. Write code to train the model for 20 additional epochs and write down what train and test accuracy you obtain.
- e. ToDo #4. Now practice using the OO functional API of keras by constructing your own functional model, modelDFNN2.
 - i. To simplify the task, create this model so that it uses the same architecture as the sequential model.
 - ii. For the final Dense layer use activation="softmax"
 - iii. Now to gain experience invoking methods on objects you have created, write code to compile(), fit() your new model on the training data.
 - iv. Then write code to evaluate your new model on the test data.
- f. ToDo #5. Write down what train and test accuracy do you observe? How does it compare to the sequential model?
- g. For fun: explore the architecture space by adding or removing layers from your model, retrain from scratch. Learn how that impacts model performance. State of the art accuracy is around 99.97% . You do not need to achieve that (nor is that expected here) but to give you an idea.

Part 2: Open and explore the code in “convnet.ipynb” then extend it as described below.

- Part 2 of this exercise uses the CIFAR10 dataset.
- CIFAR10 has 50,000 training images and 10,000 test images. Each image is a 32x32 pixel RGB (3 channel) image.
- Each image is one of 10 classes

| | | | | | | | | | | | |
|------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| airplane |  |  |  |  |  |  |  |  |  |  |  |
| automobile |  |  |  |  |  |  |  |  |  |  |  |
| bird |  |  |  |  |  |  |  |  |  |  |  |
| cat |  |  |  |  |  |  |  |  |  |  |  |
| deer |  |  |  |  |  |  |  |  |  |  |  |
| dog |  |  |  |  |  |  |  |  |  |  |  |
| frog |  |  |  |  |  |  |  |  |  |  |  |
| horse |  |  |  |  |  |  |  |  |  |  |  |
| ship |  |  |  |  |  |  |  |  |  |  |  |
| truck |  |  |  |  |  |  |  |  |  |  |  |
- The challenge is to build a classifier that reads the image and predicts which class of object it contains.
- It has been observed that human accuracy is about 94% (A. Karpathy)

<http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/>

 - d. The file contains a sequential model and a functional model to address this challenge. You should run these models to get a sense for what they do, the training time and their performance. For modelFunc you should observe:
 - a. a training loss and accuracy of around 0.33 and 88% respectively
 - b. a test loss and accuracy of around 0.88 and 72% respectively
 - e. Now **to help you practice OOP in python, your task is to construct a new class (type) called MyModelSubClass which is a keras.Model.**
 - a. Inside your new class:
 - i. define a constructor method (__init__) which defines class attributes which are the layers to be used later in the forward pass.
 - ii. define the forward pass method (call) which takes an input tensor (inputs) and the training Boolean as input and then passes inputs through the layers of the model in the appropriate order. This successively transforms the inputs into the outputs. Then return the outputs.

- iii. define this method to force the definition of the shapes of all the layers of your new class

```
def model(self):
```

```
    x = keras.Input(shape=(32, 32, 3))
```

```
    return keras.Model(inputs=[x], outputs=self.call(x))
```

- b. Outside your class:

- i. instantiate your model:

1. model3=MyModelSubClass().model()

- ii. print the size and shape of each layer via

1. print(model3.summary())

2. If you don't see the following (note your names could be different, but not the output shape and Param count), then find and fix your bug:

```
Model: "functional_3"
```

| Layer (type) | Output Shape | Param # |
|---------------------------|-----------------------|---------|
| ===== | | |
| input_4 (InputLayer) | [(None, 32, 32, 3)] | 0 |
| ----- | | |
| conv0 (Conv2D) | (None, 32, 32, 32) | 896 |
| ----- | | |
| bn0 (BatchNormalization) | (None, 32, 32, 32) | 128 |
| ----- | | |
| relu0 (ReLU) | (None, 32, 32, 32) | 0 |
| ----- | | |
| mp0 (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| ----- | | |
| conv1 (Conv2D) | (None, 16, 16, 64) | 18496 |
| ----- | | |
| bn1 (BatchNormalization) | (None, 16, 16, 64) | 256 |
| ----- | | |
| relu1 (ReLU) | (None, 16, 16, 64) | 0 |
| ----- | | |
| mp1 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| ----- | | |
| conv2 (Conv2D) | (None, 8, 8, 128) | 73856 |
| ----- | | |
| bn2 (BatchNormalization) | (None, 8, 8, 128) | 512 |
| ----- | | |
| relu2 (ReLU) | (None, 8, 8, 128) | 0 |
| ----- | | |
| flatten (Flatten) | (None, 8192) | 0 |
| ----- | | |
| dense1 (Dense) | (None, 64) | 524352 |
| ----- | | |
| denseOut (Dense) | (None, 10) | 650 |
| ===== | | |
| Total params: 619,146 | | |
| Trainable params: 618,698 | | |
| Non-trainable params: 448 | | |
| ----- | | |
| None | | |

- c. Compile your model

- d. Fit your model

- e. Evaluate your model on the test data.

- f. Now explore

- a. See what kind of accuracy you can get by training longer, increasing model size, changing the kernel sizes.