Practicum (Bonus): Intro to Image Processing & Tips in Deep Learning

Started: Apr 29 at 8:03pm

Quiz Instructions

Practicum: Intro to Image Processing & Tips in Deep Learning

This practice aims to explore different techniques in Deep Learning that can be applied to improve feature learning.

- This in-class hands-on activity reflects the key contents in Chapter 4 and Chapter 11 of the textbook and the following course slides:

Lecture15_Neural_Network.pdf (https://canvas.slu.edu/courses/68524/files/6244207?wrap=1)

Lecture16_Softmax_multi-class_Keras.pdf (https://canvas.slu.edu/courses/68524/files/6264733?wrap=1)

- We suggest everyone practice the codes and submit the results.
- You can always leave this survey anytime without clicking submit button. Canvas will automatically save your work for latter use.

This practicum serves 40 extra points added to section 'Labs'.

Question 1 0 pts

Practicum: Intro to Image Processing & Tips in Deep Learning using Keras/Tensorflow

This practice aims to explore image processing techniques and perform clustering analysis on the image data.

Useful Reference:

1. **https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb (https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb)**
2. **https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/ (https://machinelearningmastery.com/how-to-develop-a-**

cnn-from-scratch-for-cifar-10-photo-classification/)

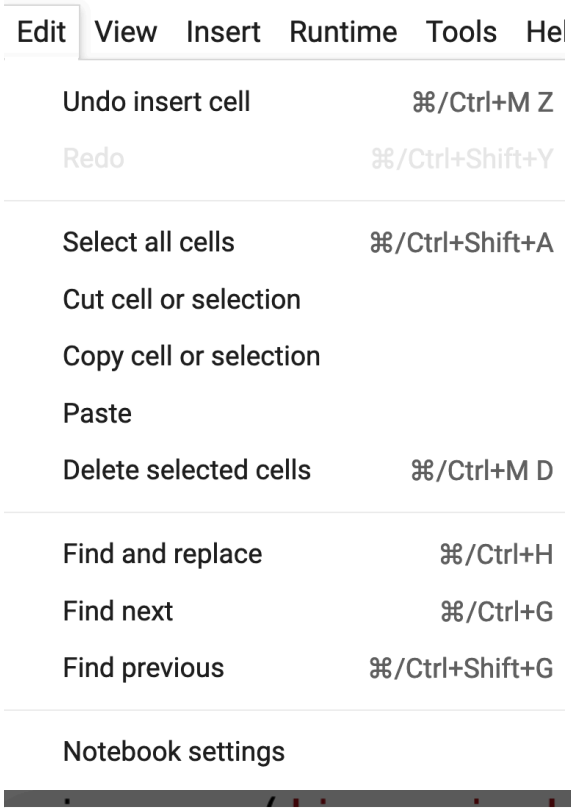
3. Google resources

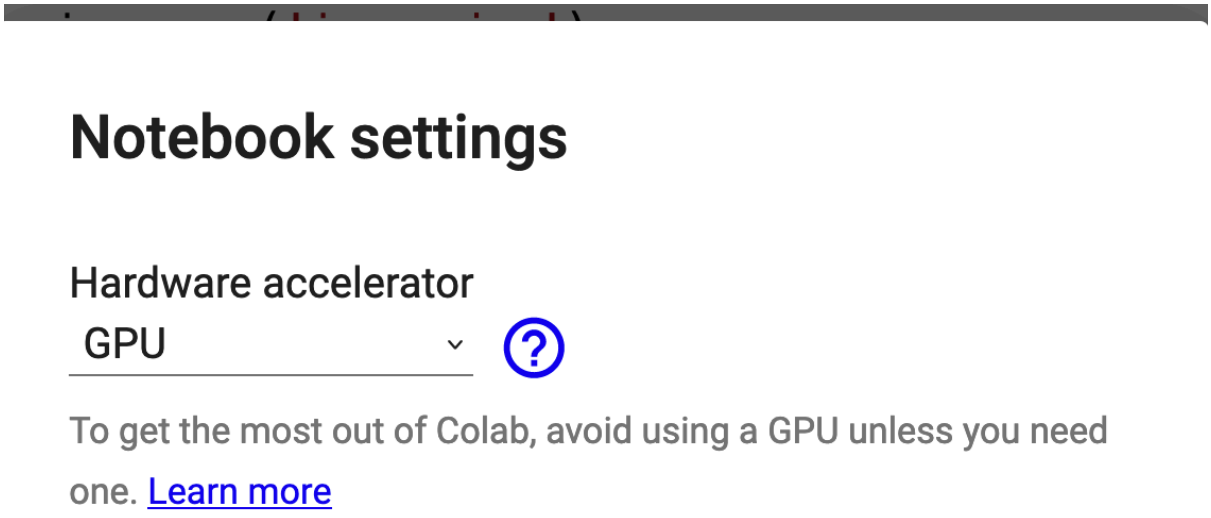
Recommended Papers:

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift(**https://arxiv.org/pdf/1502.03167.pdf (https://arxiv.org/pdf/1502.03167.pdf)**)
- Understanding the difficulty of training deep feedforward neural networks(**http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf (http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf)**)

To start this practicum, let's do all work on the Google Colab, and set the GPU as the hardware for model training accelerator

On google colab, click 'Edit' -> 'Notebook setting' -> 'Hardware accelerator' -> choose GPU





Edit View Insert Format Tools Table

12pt Paragraph B I U A T²

p

0 words

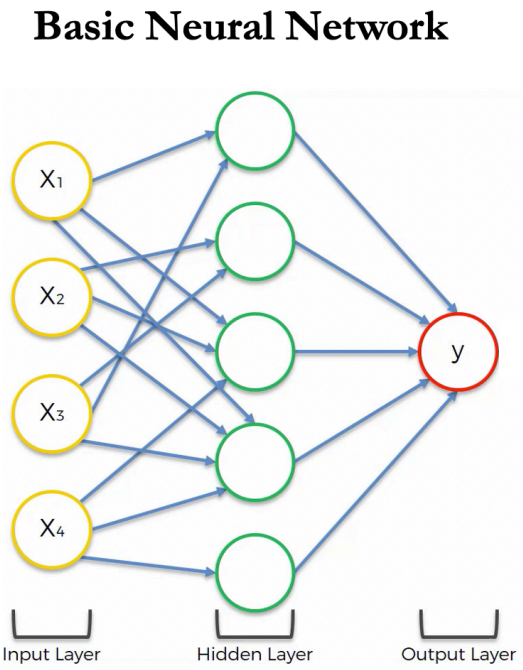
Question 2 1 pts

Slide 1

Multi-layer neural network

❖ Multi-layer neural network is composed of the three major components:

1. **Input layer:**
   each circle represents each of input features X
2. **One or more layers of hidden layers:**
   each layer contains neurons, each neuron is a weighted sum of inputs $\theta^T \cdot X$, and apply specific activation function
3. **One output layer:**
   defines the predictions, either regression outputs, or binary outputs, or multiple outputs for classification

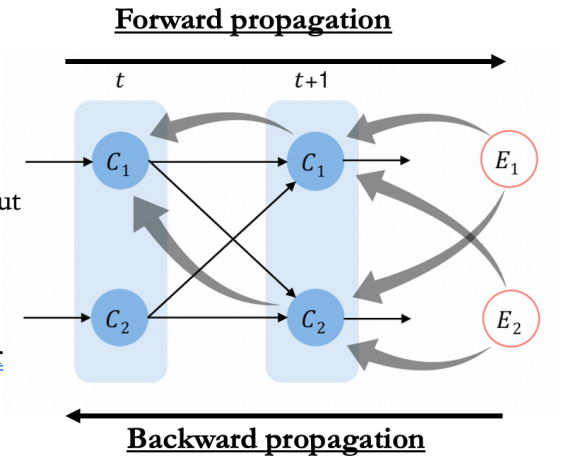


Slide 2

Neural Network Learning

- **Neural Network Learning: Two Processes**
  - Forward propagation: present an example (data) into neural network. Compute activation into units and output from units.
  - Backward propagation: propagate error back from output layer to the input layer and compute derivatives (or gradients).

- **Backward propagation**
  - ✓ Adjust weights using known examples (training data) (x_1,x_2,x_3,…,x_d,target).
  - ✓ Try to adjust weights so that the difference between the output of the neural network *y* and t (target) becomes smaller and smaller. (adjust using $\theta^{i} = \theta^{i-1} - \eta * \frac{dL(\theta)}{d\theta}$)
  - ✓ Goal is to minimize Error (difference) as we did for one layer neural network using gradient descent.

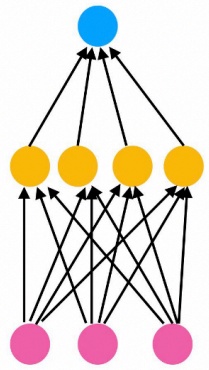


Slide 3

Training an MLP with Keras- an open-source deep learning library

TensorFlow

Keras



https://keras.io/

Slide 4

Next: Create a Keras model

Step 1: Envision the architecture

- Number of layers?
- Number of hidden nodes?
- Hidden activation functions?
- Output activation functions?

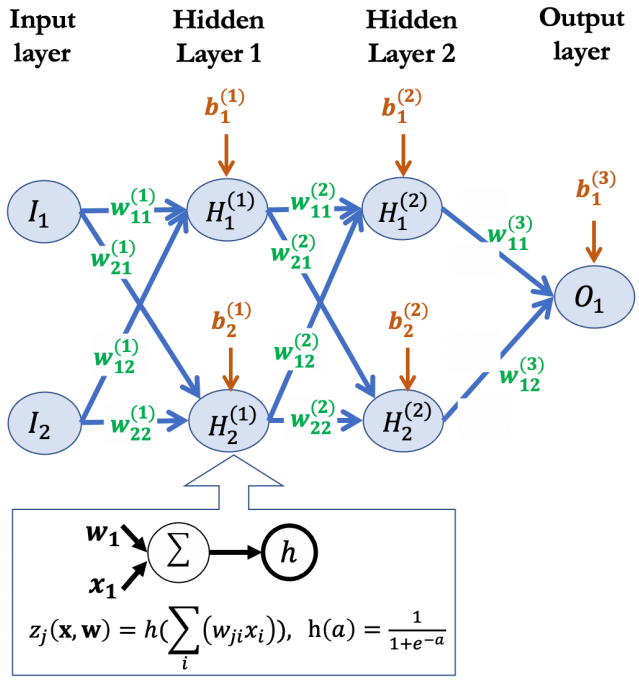
Step 2: Define the model architecture

Step 3: Compile the model

- Choose loss function (i.e., MSE)
- Define optimization algorithms (i.e.,GD)

Step 4: Fit the model

- Input the training features and labels
- Forward/Backward propagation
- Update weights using gradient descent
- Optimize loss function



Step 5: Make prediction

Build a three-layer neural network with sigmoid function as output

Edit View Insert Format Tools Table

12pt Paragraph B I U A T²

p 0 words </>

Slide 5

Practice dataset I: binary classification

In this exercise, we will use the SPECT Heart dataset (https://archive.ics.uci.edu/ml/datasets/SPECT+Heart) to fit a classification tree for cardiac diagnosis using functions in the 'sklearn' module.

SPECT Heart Data Set

Download: Data Folder, Data Set Description

Abstract: Data on cardiac Single Proton Emission Computed Tomography (SPECT) images. Each patient classified into two categories: normal and abnormal.

Data Set Characteristics:	Multivariate	Number of Instances:	267	Area:	Life
Attribute Characteristics:	Categorical	Number of Attributes:	22	Date Donated	2001-10-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	206366

https://archive.ics.uci.edu/ml/datasets/spect+heart

Slide 6

Practice dataset

Task1: (5 points) Prepare the dataset

Requirement: Download the dataset from the UCI website: https://archive.ics.uci.edu/ml/datasets/SPECT+Heart . The data are saved in file 'SPECT.train' and 'SPECT.test', which are the csv format as shown in the figure below. You can also find the description of this data on the homepage of this dataset.

The dataset contains 22 input attributes extracted from the image. The target variable (first column) contains two classes for diagnosis: normal and abnormal.

- Parent Directory
- DonorNote.txt
- SPECT.names
- SPECT.test
- SPECT.train
- SPECTF.names
- SPECTF.test
- SPECTF.train
- SPECTFincorrect.test

```
1  1,0,0,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0
2  1,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,1
3  1,1,0,1,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0
4  1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1
5  1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,0,0,0,0
6  1,0,0,0,1,0,0,0,0,1,0,0,0,1,1,0,1,0,0,0,1,0,1
7  1,1,0,1,1,0,0,0,1,0,1,0,1,1,0,0,0,0,0,0,0,1,1
8  1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1
9  1,0,0,1,0,0,0,1,1,0,0,0,0,1,0,1,0,0,0,0,0,1,1
10 1,0,1,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0
```

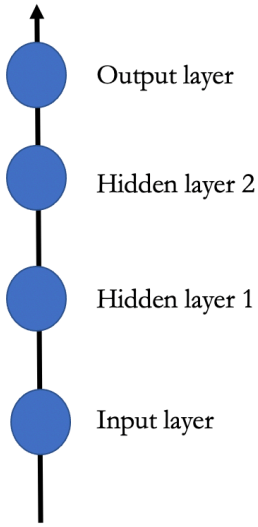
Slide 7

Create a Keras model

❑ Build Typical Deep Multilayer Neural Net Architecture in Keras

Step 1: Envision the architecture

- Number of layers? (Two hidden layers, one output layer)
- Number of hidden nodes? (100 hidden nodes)
- Hidden activation functions? (relu)
- Output activation functions? (sigmoid)



Plan: build a three-layer neural network with binary function as output

```
from keras.models import Sequential
model = Sequential() # create Sequential model
model.add() # input layer -> hidden layer 1
model.add() # hidden layer 1 -> hidden layer 2
model.add() # hidden layer 2 -> output layer
```

Data link: https://archive.ics.uci.edu/ml/datasets/spect+heart (https://archive.ics.uci.edu/ml/datasets/spect+heart)

Question 3 1 pts

Practice 1: Build a multi-layer neural network for binary classification

Following our steps below to import the data as follows:

```
# practice 1.1: Load training data
import pandas as pd
from sklearn.utils import shuffle
train_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine
-learning-databases/spect/SPECT.train', header=None)
train_data.columns = ['Label']+[ 'F'+str(i) for i in range(1,2
3)]
```

```
train_data = shuffle(train_data) # shuffle data
print(train_data)
```

```
# practice 1.2: Load testing data
import pandas as pd
test_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/spect/SPECT.test', header=None)
test_data.columns = ['Label']+[ 'F'+str(i) for i in range(1,23)]
print(test_data)
```

```
# practice 1.3: import Sequential class and Dense class from ker as
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# practice 1.4: build a three-layer neural network with binary f unction as output
model = Sequential()
model.add()
```

(optional, but recommended) Task 1: Run Practice 1.1 & 1.2 to load the dataset, and print the shape of the dataset

(optional, but recommended) Task 2: Run the above codes of Practice 1.3 & 1.4 , and report what errors you get?

(optional, but recommended) Task 3: Describe what we should add to resolve the errors.

Edit View Insert Format Tools Table

12pt Paragraph B I U A T²

p 0 words </>

Question 4 1 pts

Slide 8

Create a Keras model

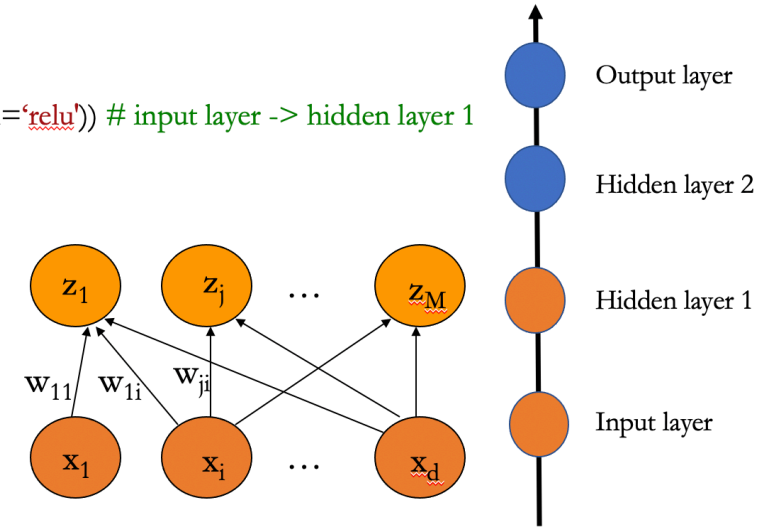
Step 2: Define the model architecture

✓ Hidden layer 1: 100 neurons, relu function

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential() # initialize Sequential model
# add connection layers into sequential model
model.add(Dense(100, input_shape=(22,),activation='relu')) # input layer -> hidden layer 1
model.add() # hidden layer 1 -> hidden layer 2
model.add() # hidden layer 2 -> output layer
```

Dimension of output space

Shape of input for first layer



Details in document: https://keras.io/layers/core/#dense

Edit View Insert Format Tools Table

12pt Paragraph B I U A T²

p 0 words </>

Question 5 1 pts

Practice 2: Define the first hidden layer in model architecture

Following steps in Practice 1, and add the layer configuration in keras,

First, we need to know how many features we have in the training data, this will determine the number of input neurons we need.

```
## Practice 2.1: extract features and labels from the training/test data
train_label = train_data['Label']
train_features = train_data.drop('Label',axis= 1)
print("train_features.shape:",train_features.shape)

test_label = test_data['Label']
test_features = test_data.drop('Label',axis=1)
print("test_features.shape:",test_features.shape)
```

```
## Practice 2.2: Define the first hidden layer in model architecture
from keras.models import Sequential
from keras.layers import Dense
model = Sequential() # initialize Sequential model

# add connection layers into sequential model
model.add(Dense(100, input_shape=(22,),activation='relu')) # input layer -> hidden layer 1

model.summary()
```

```
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_1 (Dense)              (None, 100)               2300

=================================================================
Total params: 2,300
Trainable params: 2,300
Non-trainable params: 0
_________________________________________________________________
```

(optional, but recommended) Task 1: Run the above codes, and describe how many parameters are in total in this network after we add the dense layer.
(optional, but recommended) Task 2: Could you describe how the number of parameters in this added layer is calculated?

Edit View Insert Format Tools Table

12pt Paragraph

p 0 words

Question 6 1 pts

Practice 3: Add more hidden layers and output layers into network

Following the steps in Practice 2, and adding the additional layers to model,

```
## Practice 3: Add more hidden layers and output layers into net
work
from keras.models import Sequential
from keras.layers import Dense

model = Sequential() # initialize Sequential model
model.add(Dense(100, input_shape=(22,), activation='relu')) # in
put layer -> hidden layer 1
model.add(Dense(100, activation='relu')) # hidden layer 1 -> hid
den layer 2
model.add(Dense(1, activation='sigmoid')) # hidden layer 2 -> ou
tput layer
model.summary()
```

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_2 (Dense)              (None, 100)               2300

 dense_3 (Dense)              (None, 100)               10100

 dense_4 (Dense)              (None, 1)                 101

=================================================================
Total params: 12,501
Trainable params: 12,501
Non-trainable params: 0
```

(optional, but recommended) Task 1: Run the above codes, and describe how many parameters are in total in this network after we add the new dense layers.

(optional, but recommended) Task 2: Also, could you describe how the number of parameters in every hidden layer is calculated?

Practice 4: Compile the model for backpropagation, and start training the model on data

Following the steps in Practice 4, we need to configure the loss function and proper gradient descent algorithm

```
## Practice 4.1: Compile the model for backpropagation, and start training model on data
from keras.models import Sequential
from keras.layers import Dense

model = Sequential() # initialize Sequential model
model.add(Dense(100, input_shape=(22,), activation='relu')) # input layer -> hidden layer 1
model.add(Dense(100, activation='relu')) # hidden layer 1 -> hidden layer 2
model.add(Dense(1, activation='sigmoid')) # hidden layer 2 -> output layer

# Compile model, use binary crossentropy, and stochastic gradient descent for optimization
model.compile(loss='binary_crossentropy', optimizer='SGD', metrics=['accuracy'])

model.summary()
```

```
## Practice 4.2: start training the model on the features and labels, set the training epochs to 10
model.fit(train_features, train_label, epochs = 10)
```

```
model.fit(train_features, train_label, epochs = 20)
```

```
Epoch 1/20
3/3 [==============================] - 0s 4ms/step - loss: 0.6142 - accuracy: 0.6625
Epoch 2/20
3/3 [==============================] - 0s 5ms/step - loss: 0.6130 - accuracy: 0.6625
Epoch 3/20
3/3 [==============================] - 0s 4ms/step - loss: 0.6118 - accuracy: 0.6625
Epoch 4/20
3/3 [==============================] - 0s 4ms/step - loss: 0.6107 - accuracy: 0.6625
Epoch 5/20
3/3 [==============================] - 0s 5ms/step - loss: 0.6097 - accuracy: 0.6625
Epoch 6/20
3/3 [==============================] - 0s 3ms/step - loss: 0.6085 - accuracy: 0.6875
Epoch 7/20
3/3 [==============================] - 0s 5ms/step - loss: 0.6075 - accuracy: 0.6875
```

(optional, but recommended) Task: Run the above codes, increase the number of epochs to see whether the accuracy gets improved, or loss gets decreased.

Question 7 1 pts

Practice 5: Evaluate the neural network

Following the steps in Practice 5, we can save the model to a local disk, or load the model for predictions.

```
## practice 5.1: save the model to local disk, or load the model for predictions
from tensorflow.keras.models import load_model
model.save("SPECT.h5")
model_loaded = load_model("SPECT.h5")
```

```
## practice 5.2: evaluate the model performance using evaluate() method
model_loaded.evaluate(train_features, train_label)
```

We can also make a prediction on training and testing set. And using sklearn's functions for evaluation

```
## practice 5.3: make a prediction on training set and test set
train_predictions = model_loaded.predict(train_features)
test_predictions = model_loaded.predict(test_features)
```

```
print("train_predictions: ",train_predictions) # the prediction
is the probability of the class
```

```
## practice 5.4: convert probability to labels
import numpy as np
train_prediction_labels = (train_predictions > 0.5).astype(int)
test_prediction_labels = (test_predictions > 0.5).astype(int)
```

```
## practice 5.5: convert probability to labels
from sklearn.metrics import accuracy_score
print("Training accuracy: ", accuracy_score(train_label, train_p
rediction_labels))
print("Test accuracy: ", accuracy_score(test_label, test_predict
ion_labels))
```

(optional, but recommended) Task 1: Run **practice 5.1,** and check if the model file is saved in your local folder

(optional, but recommended) Task 2: Run **practice 5.2 & 5.3 & 5.4,** and check how the predicted probability is converted into predicted labels

(optional, but recommended) Task 3: Run codes of **practice 5.2** and **practice 5.5**, and check if the accuracy results are same.

(optional, but recommended) Task 4: practice how to calculate f1-score, precision, and recall using sklearn functions

Question 8 1 pts

Practice 6: Compare the performance between Keras' model with sklearn's model.

Please use sklearn's neural network function to build a neural network model for the same dataset.

Link: **https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)**

sklearn.neural_network.MLPClassifier

class `sklearn.neural_network.`**`MLPClassifier`**(*hidden_layer_sizes=(100,)*, *activation='relu'*, ***, *solver='adam'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*, *n_iter_no_change=10*, *max_fun=15000*) [source]

Multi-layer Perceptron classifier.

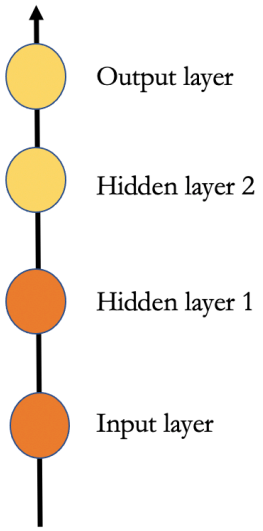
Task 1: Define proper hyper-parameters in sklearn.neural_network.MLPClassifier for the same architecture in Figure below:

Create a Keras model

Step 4: Verify the model architecture

```
model.summary()
```

```
Model: "sequential_43"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_100 (Dense)            (None, 100)               2300
_________________________________________________________________
dense_101 (Dense)            (None, 100)               10100
_________________________________________________________________
dense_102 (Dense)            (None, 1)                 101
=================================================================
Total params: 12,501
Trainable params: 12,501
Non-trainable params: 0
_________________________________________________________________
```



How is this calculated?

Details in document: https://keras.io/models/model/#fit

Task 2: Train the sklearn model on the following training data, and report the accuracy/f1-score on the test data.

```
# practice 6.1: Load training data
import pandas as pd
from sklearn.utils import shuffle
train_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine
-learning-databases/spect/SPECT.train', header=None)
train_data.columns = ['Label']+[ 'F'+str(i) for i in range(1,2
3)]
train_data = shuffle(train_data) # shuffle data
train_data
```

```
# practice 6.2: Load testing data
import pandas as pd
test_data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/spect/SPECT.test', header=None)
test_data.columns = ['Label']+[ 'F'+str(i) for i in range(1,23)]
test_data
```

```
## Practice 6.3: extract features and labels from the training/test data
train_label = train_data['Label']
train_features = train_data.drop('Label',axis= 1)
print("train_features.shape:",train_features.shape)

test_label = test_data['Label']
test_features = test_data.drop('Label',axis=1)
print("test_features.shape:",test_features.shape)
```

(optional, but recommended) Task 1: Compare your results from sklearn model and the results from Keras model, describe the difference in results.

Slide 1

Practice dataset II: multi-class classification

❖ Build Typical Deep Multilayer Neural Net Architecture in Keras

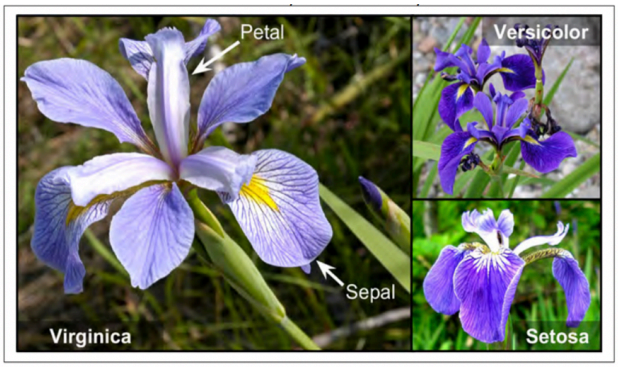
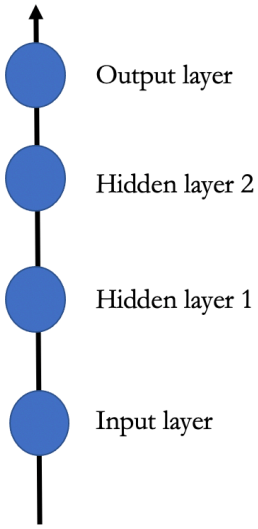


Figure 4-22. Flowers of three iris plant species[16]

```
from keras.models import Sequential
model = Sequential() # create Sequential model
model.add() # input layer -> hidden layer 1
model.add() # hidden layer 1 -> hidden layer 2
model.add() # hidden layer 2 -> output layer
```

Plan: build a three-layer neural network with softmax function as output



Question 9 1 pts

Practice 7: Build a three-layer neural network with softmax function for multi-class classification

```
# practice 7.1: setup iris dataset for multi-class classificatio
n
from sklearn import datasets
import numpy as np

iris = datasets.load_iris()

# get features from iris data
X = iris["data"][:, :] # using all features

# get binary labels for binary classification
y = iris["target"]

print(X.shape)
print(y)
```

```
# practice 7.2: Define the neural network classifier with softma
x function as output
from keras.models import Sequential
from keras.layers import Dense

model = Sequential() # initialize Sequential model
model.add(Dense(10, input_shape=(X.shape[1],), activation='rel
u')) # input layer -> hidden layer 1
model.add(Dense(5, activation='relu')) # hidden layer 1 -> hidde
n layer 2
model.add(Dense(3, activation='softmax')) # hidden layer 2 -> ou
tput layer (3 classes, use softmax function to calculate probabi
lity)
```

```
# Compile model, use categorical_crossentropy as loss function f
or softmax function
model.compile(loss='categorical_crossentropy',
optimizer='SGD',
metrics=['accuracy'])
```

```
# practice 7.3: Train the model on the training set
model.fit(X, y, validation_split = 0.2, epochs=10)
```

Task 1: Run the above codes, and report the errors you see.

Hint: we are missing one step, that we should convert labels into one-hot encoding labels for calculating categorical crossentropy loss

```
## re-fit model on the training features and one-hot encoding la
bels
import tensorflow as tf
y_onehot = tf.keras.utils.to_categorical(y, num_classes=3, dtype
='float32')
print(y_onehot)
model.fit(X, y_onehot, validation_split = 0.2, epochs=10)
```

```
model.fit(X, y_onehot, validation_split = 0.2, epochs=10)
```

```
Epoch 1/10
4/4 [==============================] - 0s 131ms/step - loss: 1.0934 - accuracy: 0.1667 - val_loss: 0.7401 - val_accuracy: 1.0000
Epoch 2/10
4/4 [==============================] - 0s 15ms/step - loss: 1.0761 - accuracy: 0.1667 - val_loss: 0.7799 - val_accuracy: 1.0000
Epoch 3/10
4/4 [==============================] - 0s 16ms/step - loss: 1.0637 - accuracy: 0.1667 - val_loss: 0.8122 - val_accuracy: 1.0000
Epoch 4/10
4/4 [==============================] - 0s 20ms/step - loss: 1.0531 - accuracy: 0.1833 - val_loss: 0.8431 - val_accuracy: 1.0000
Epoch 5/10
4/4 [==============================] - 0s 28ms/step - loss: 1.0439 - accuracy: 0.2000 - val_loss: 0.8674 - val_accuracy: 1.0000
```

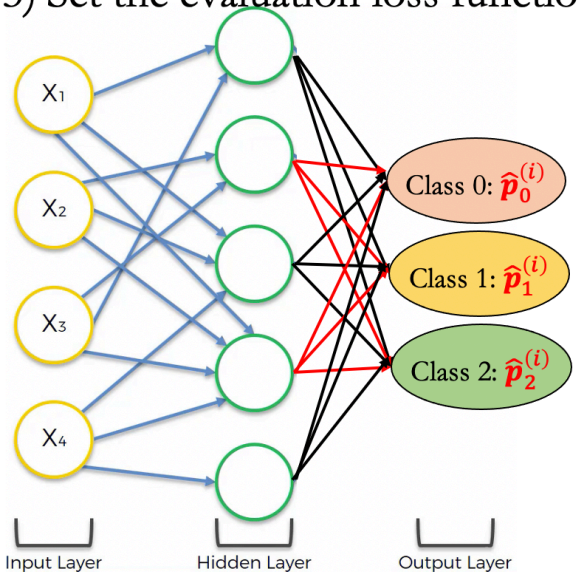
Task 2: Fix the error in Task 1, and report the training process.

Slide 23

How does Neural Network resolve Multi-class Classification

➢ **For Multi-layer Neural Network Classification Prediction (Multi-class)**

1) Set the number of output neurons to K classes
2) Set the activation function of the neuron in last layer to SoftMax Function
3) Set the evaluation loss function to Cross-Entropy



$$\text{Loss} = \text{Cross-Entropy} = \frac{1}{N}\sum_{i=1}^{N} -\sum_{k=1}^{K} [y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)]$$

Slide 24

Slide 25

How SoftMax Regression makes its classification decision

❑ **Equation: Softmax function**

$$\begin{cases} a_k = \sum_{j=0}^{d} w_{1j}\, x_j \\ \mathbf{P}(\text{Class } \boldsymbol{k}|\mathbf{x}) = \dfrac{\exp(a_k)}{\sum_{k=1}^{C} \exp(a_k)} \end{cases}$$

$$\hat{y} = \underset{k}{\operatorname{argmax}}\, \mathbf{P}(\text{Class } \boldsymbol{k}|\mathbf{x})$$

Note 1: Softmax Regression predicts the class with the highest probability

Note 2: Softmax Regression is applied on exclusive classes. Shouldn't be applied on multi-label classification

```
softmax_regression.predict_proba(X)
```

```
array([[9.81801790e-01, 1.81981959e-02, 1.43556907e-08],
       [9.71727348e-01, 2.82726221e-02, 3.00307256e-08],
       [9.85451821e-01, 1.45481667e-02, 1.22669829e-08],
       [9.76299555e-01, 2.37004051e-02, 3.95392775e-08],
       [9.85388794e-01, 1.46111946e-02, 1.18844386e-08],
       [9.70470231e-01, 2.95296954e-02, 7.31658657e-08],
       [9.86892099e-01, 1.31078812e-02, 1.98813481e-08],
       [9.76403999e-01, 2.35959736e-02, 2.74987214e-08],
       [9.79846470e-01, 2.01534997e-02, 3.05483138e-08],
       [9.69136281e-01, 3.08636877e-02, 3.15045384e-08],
       [9.76507990e-01, 2.34919910e-02, 1.91247633e-08],
       [9.75414682e-01, 2.45852747e-02, 4.36441157e-08],
       [9.74561901e-01, 2.54380777e-02, 2.13874589e-08],
       [9.91974148e-01, 8.02584827e-03, 3.87570298e-09],
```

$\mathbf{P}(\text{Class } 0|\mathbf{x})$ $\mathbf{P}(\text{Class } 1|\mathbf{x})$ $\mathbf{P}(\text{Class } 2|\mathbf{x})$

Question 10 1 pts

Practice 7: Monitor the model performance using validation set

Keras provides the option for us to monitor the training performance using validation set. We can use validation set in two ways.

First, we can split the whole data into training and validation set using sklearn's function

```
# Practice 7.1: Monitor the model performance using validation s
et
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(train_feature
s, train_label, test_size=0.2, random_state=42)

## option 1: monitor the model training using validation data
```

```
model.fit(X_train, y_train, validation_data= (X_val, y_val), epo
chs = 20)
```

```
## monitor the model training using validation data
model.fit(X_train, y_train,  validation_data= (X_val, y_val), epochs = 20)
```

```
Epoch 1/20
2/2 [==============================] - 0s 121ms/step - loss: 0.6099 - accuracy: 0.7031 - val_loss: 0.6052 - val_accuracy: 0.7500
Epoch 2/20
2/2 [==============================] - 0s 52ms/step - loss: 0.6091 - accuracy: 0.7031 - val_loss: 0.6047 - val_accuracy: 0.7500
Epoch 3/20
2/2 [==============================] - 0s 41ms/step - loss: 0.6083 - accuracy: 0.7031 - val_loss: 0.6041 - val_accuracy: 0.7500
Epoch 4/20
2/2 [==============================] - 0s 56ms/step - loss: 0.6077 - accuracy: 0.7031 - val_loss: 0.6035 - val_accuracy: 0.7500
Epoch 5/20
```

Second, we can directly split data inside model.fit() method

```
## option 2: monitor the model training using validation data
model.fit(X_train, y_train, validation_split=0.2, epochs = 20)
```

Task: Practice the codes, and report the training process using a validation set

Question 11 1 pts

Problem I (25 points, mandatory to complete): Apply all above analysis to build Keras model for multi-class classification on Iris dataset

Task1: (16 points) Build a neural network using Keras (with at least two hidden layers).

Only features and labels in the training data should be used.

Requirement:

- Correctly define **the number of nodes** for the hidden layers and output layer; **(2 points)**
- Correctly define **activation functions** for the hidden layers and output layer; **(2 points)**
- Correctly define **the loss function** in 'model.compile' (e.g., loss = "binary_crossentropy"); **(2 points)**
- Correctly define **the optimizer function** in 'model.compile' (e.g., optimizer = "sgd"); **(2 points)**
- Correctly define **batch size** in 'model.fit' (e.g., batch_size=5);**(2 points)**
- Correctly use **argument 'validation_split=0.1'** in 'model.fit' to monitor the training using validation set; **(2 points)**
- Correctly define an **epoch** in 'model.fit' (e.g., epochs = 30); **(2 points)**
- Correctly use **training data and labels** in 'model.fit'; **(2 points)**

Example: train_history = model.fit(X_train, y_train_labels , validation_split=0.1, batch_size=5, epochs = 30)

Note: variable 'train_history' will hold all training history information for visualization in Task 8

If everything is set correctly, you are supposed to get similar outputs as:

```
39/39 [==============================] - 0s 1ms/step - loss: 1.5204 - accuracy: 0.4167 - val_loss: 1.2883 - val_accuracy: 0.5455
Epoch 24/30
39/39 [==============================] - 0s 1ms/step - loss: 1.5143 - accuracy: 0.4167 - val_loss: 1.2810 - val_accuracy: 0.5455
Epoch 25/30
39/39 [==============================] - 0s 1ms/step - loss: 1.5084 - accuracy: 0.4167 - val_loss: 1.2705 - val_accuracy: 0.5455
Epoch 26/30
39/39 [==============================] - 0s 1ms/step - loss: 1.5027 - accuracy: 0.4167 - val_loss: 1.2648 - val_accuracy: 0.5909
Epoch 27/30
39/39 [==============================] - 0s 1ms/step - loss: 1.4977 - accuracy: 0.4167 - val_loss: 1.2563 - val_accuracy: 0.5909
Epoch 28/30
39/39 [==============================] - 0s 1ms/step - loss: 1.4920 - accuracy: 0.4167 - val_loss: 1.2508 - val_accuracy: 0.5909
Epoch 29/30
39/39 [==============================] - 0s 1ms/step - loss: 1.4865 - accuracy: 0.4219 - val_loss: 1.2463 - val_accuracy: 0.5909
Epoch 30/30
39/39 [==============================] - 0s 2ms/step - loss: 1.4815 - accuracy: 0.4219 - val_loss: 1.2383 - val_accuracy: 0.6364
```

Task 2: (4 points) Report the accuracy of the training and test dataset.

Hint: model.evaluate() in Keras can be used here

training data

```
model.evaluate(X_train, y_train_labels)

# testing data
model.evaluate(X_test,y_test_labels)
```

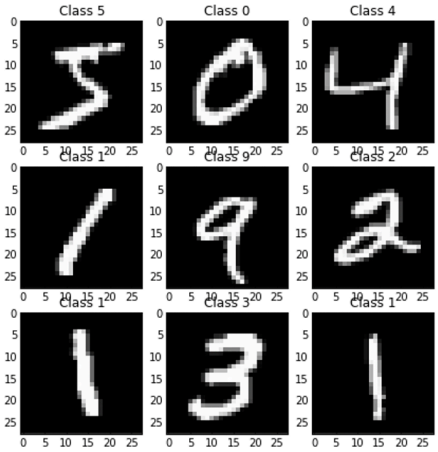
Task 3: (5 points) Increase model complexity by adding more hidden layers and nodes (i.e., 3 hidden layers, 100 hidden nodes, epoch 50). Any improvement on validation or test set?

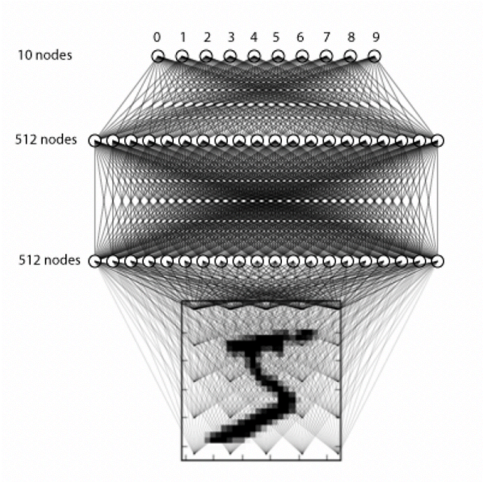
Part II: Apply Neural Network on Images

Work on Images

❖ **Work through this Keras Jupyter notebook for MNIST.**

- 28 ×28 pixel grid
- Flatten into 784 ×1





https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb

Demo link:

https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb (https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb)

Question 12 8 pts

Step 1: Basic image processing in python

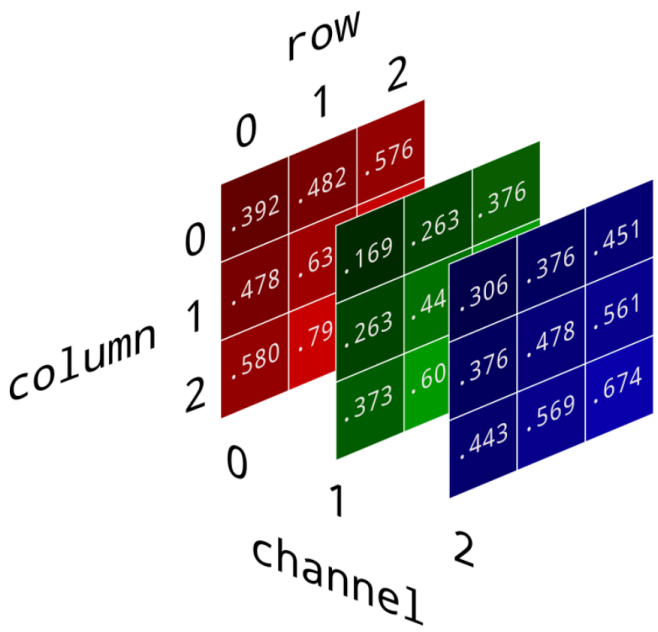
Using matplotlib, you can import an image into python from the file, save it into a variable, and visualize the data as an image. (Python will treat the image as an array of numbers)

```
# step 1.1: download an online image
!wget https://images.all-free-download.com/images/graphicthumb/airplane_landing_199029.jpg
```

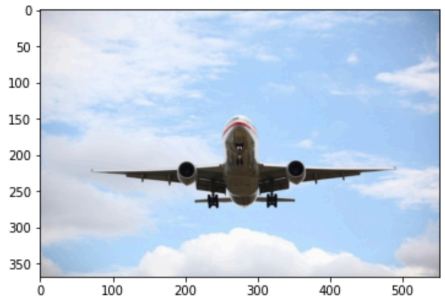
```
# step 1.2: load image into python
import matplotlib.pyplot as plt
data = plt.imread('airplane_landing_199029.jpg')
plt.imshow(data)
plt.show()
```

Note: The color image has three dimensions, where the first two dimensions correspond to the height and width of the image. The total number of pixels is height*width. The last dimension (i.e., 3) corresponds to the three color channels (red, green, and blue colors) in each pixel.

You can examine the red, green and blue data in a particular pixel in the image by setting the row index and column index.



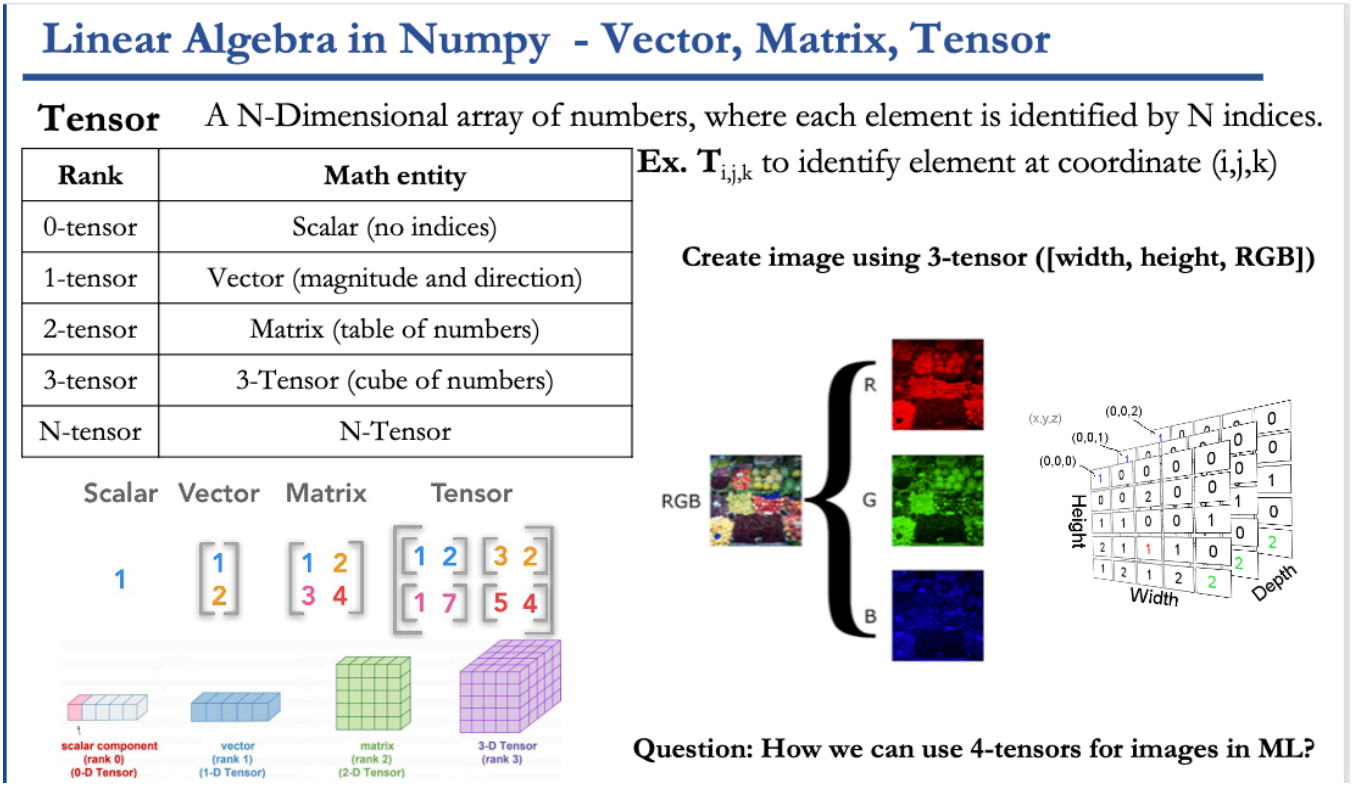
The pixel value is a single number that represents the brightness of the pixel. The possible values for a pixel range from 0 (turned off) to 255 (maximum brightness).
Question 1: print out the variable 'data', and copy/paste the matrix to this box. What does each number in this matrix represent?
Question 2: print out the data type of variable 'data' using code **print(type(data)),** which data structure does this variable have? (pandas.dataframe or numpy.ndarray?)
Question 3: print out the shape of the matrix using code **print(data.shape)**, you will see the following shape: (368,552, 3). What does each number in this shape mean?



```
print(data.shape)
```

```
(368, 552, 3)
```

Question 4: Recall our week 1 topic about different types of tensors, which type of tensor does this variable 'data' have?

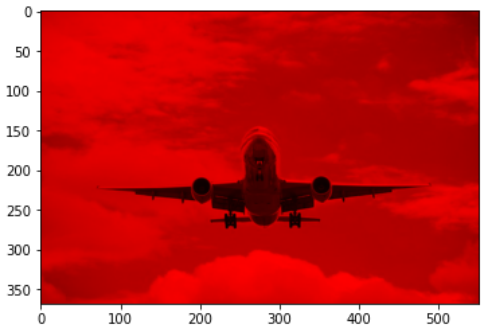


Question 13 4 pts

Step 2: After loading the image, the image data can be also modified by changing the array values

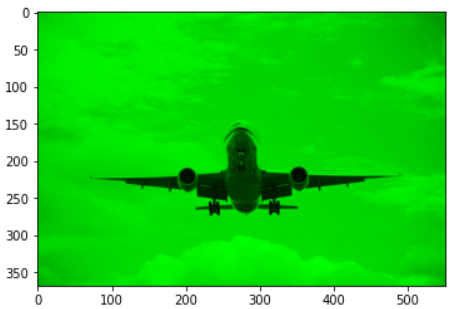
 # step 2.1: Modify the image to red channel only

```
# step 2.1: Modify the image to red channel only
data_modified = data.copy()
data_modified[:,:,1] = 0 # set intensity of green channel to 0
data_modified[:,:,2] = 0 # set intensity of blue channel to 0
plt.imshow(data_modified) # only show red channel
plt.show()
```

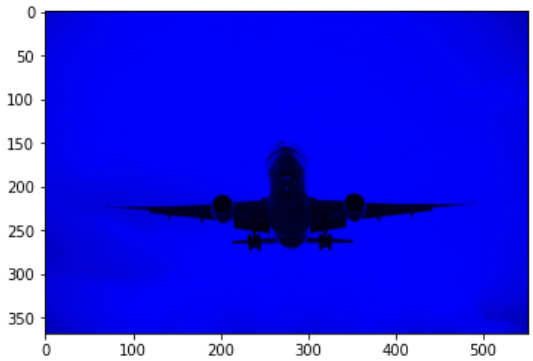


 # step 2.2: Modify the image to green channel only

```
# step 2.2: Modify the image to green channel only
data_modified = data.copy()
data_modified[:,:,0] = 0 # set intensity of red channel to 0
data_modified[:,:,2] = 0 # set intensity of blue channel to 0
plt.imshow(data_modified) # only show red channel
plt.show()
```



Task: How to modify the image to the blue channel only as follows? Provide your codes in this box.



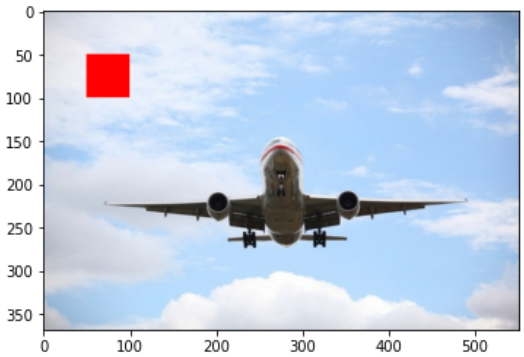
Question 14 2 pts

Step 3: You can also modify the specific regions in the image using matrix manipulation

Step 3.1: Add a red mask to the specific regions within the image

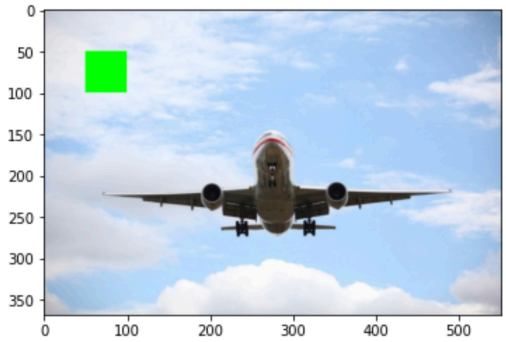
```
# step 3.1: Add red mask to the specific regions within the imag
e**
data_modified = data.copy()
```

```
data_modified[50:100,50:100,:] = [255,0,0] # change subregion to red only, make sure the order of channel colors is [red,green,blue]
plt.imshow(data_modified)
plt.show()
```



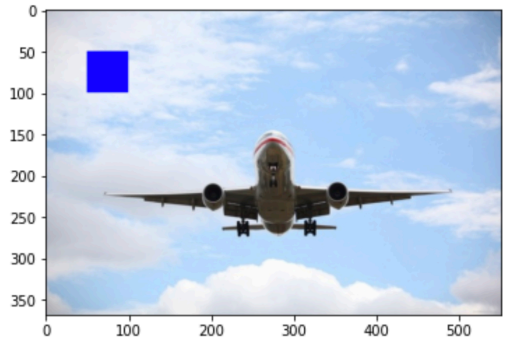
Step 3.2: Add a green mask to the specific regions within the image

```
# step 3.2: Add green mask to the specific regions within the image**
data_modified = data.copy()
data_modified[50:100,50:100,:] = [0,255,0] # change subregion to green only, make sure the order of channel colors is [red,green,blue]
plt.imshow(data_modified)
plt.show()
```



Question 1: Write your codes to add a blue mask to the specific regions within the image, and upload your image.

Sample image is below:



Question 2: You can also set the mask to other colors, replacing [255,0,0] to [215,120,234] or any other random numbers for each channel (you can get customized color at https://www.rapidtables.com/web/color/RGB_Color.html (https://www.rapidtables.com/web/color/RGB_Color.html)).

Write your codes to add a mask with different colors to the specific regions within the image, and upload your image.

Question 15 4 pts

Step 4: Apply image processing on the CIFAR10 image dataset

About CIFAR10

In addition to the MNIST dataset we have practiced in the class, we will work on another popular dataset called '**CIFAR10**', containing 32x32 color images of 10 categories of

objects. Below are labels for the objects - airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

We can directly load the dataset into python from module 'keras.datasets,' which contains:

- **Training Set:** 50000 images, dimension shape: 50000 * 32 * 32 *3
- **Test Set:** 10000 images, dimension shape: 10000 * 32 * 32 *3

```
# Step 4: Apply image processing on the CIFAR10 image dataset
from keras.datasets import cifar10
import matplotlib.pyplot as plt
import numpy as np

# load dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

labels_map = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'fro
g', 'horse', 'ship', 'truck']
```

Note: All X_train, y_train, X_test, y_test are NumPy array

Question 1: Write the following codes to print out the shape of the four data

```
print("X_train.shape: ",X_train.shape)
print("X_test.shape: ",X_test.shape)
print("y_train.shape: ",y_train.shape)
print("y_test.shape: ",y_test.shape)
```

```
X_train.shape:  (50000, 32, 32, 3)
X_test.shape:  (10000, 32, 32, 3)
y_train.shape:  (50000, 1)
y_test.shape:  (10000, 1)
```

Explain what each of the numbers in the above shape means?

- 50000:
- 10000:
- 32:
- 3:
- 1:

Question 2: Which type of tensor do the variables have?

- X_train:
- X_test:
- y_train:
- y_test:

Question 16 4 pts

Step 5: Using matplotlib, you can visualize the NumPy data as an image as follows:

```
# Step 5: Image visualization using matplotlib
fig = plt.figure(figsize=(20, 5))
for ii in range(10):
   # sample a random image from X_train
   image_indx = np.random.choice(range(len(X_train)))
   image_random = X_train[image_indx]
   image_title = labels_map[y_train[image_indx][-1]]

   # put image into subplots
   imgplot = fig.add_subplot(2,5,ii+1)
   imgplot.imshow(image_random)
   imgplot.set_title(image_title, fontsize=20)
   imgplot.axis('off')
```

Task: Upload your image to this box (do not copy/paste, use upload button to load image). You are suppose to submit similar image as below

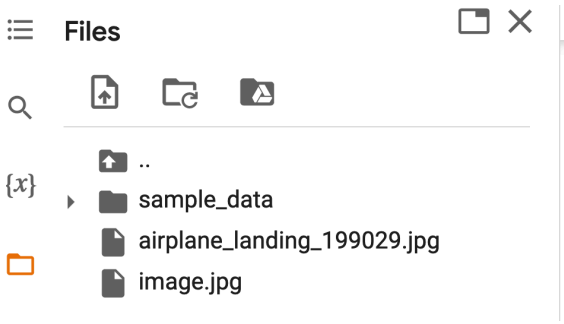
Step 6: We can also save the NumPy array into an image format using the following codes

```
# Step 6.1: sample a random image
image_index = np.random.choice(range(len(X_train)))
image_example = X_train[image_index]
```

```
# Step 6.2:  Use the Python Imaging Library 'PIL' to save image into local file
```

```
import PIL
img = PIL.Image.fromarray(X_train[image_index])
img.save('image.jpg')
```

Double-check if you have this file generated.



And then we can reload the image from the file into NumPy array and visualize the matrix using Matplotlib

```
# Step 6.3:  reload image into python
import matplotlib.pyplot as plt
data = plt.imread('image.jpg') # read the local image
plt.imshow(data)
plt.axis('off')
plt.show()
```

Task 1: If you check the shape of matrix using print(data.shape), are you getting the same shape as the original cifar image (32,32,3)?

Step 7: Application of Machine Learning algorithms in images dataset

The following contents are adapted from the tutorials in [MNIST in Keras] (**https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb (https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%20Keras.ipynb)**) and our lectures to perform the analysis.

```
## Step 7.1: Load the dataset
from keras.datasets import cifar10
(X_data, y_data), (X_test, y_test) = cifar10.load_data()
print("Training matrix shape", X_data.shape)
print("Testing matrix shape", X_test.shape)
print("y_data matrix shape", y_data.shape)
print("y_test matrix shape", y_test.shape)
```

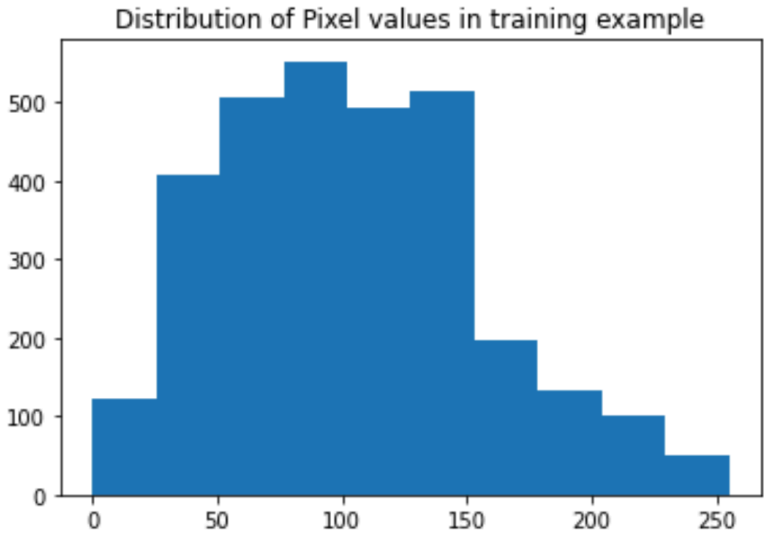
```
print("y_data: ",y_data)
```

```
(X_data, y_data), (X_test, y_test) = cifar10.load_data()
print("Training matrix shape", X_data.shape)
print("Testing matrix shape", X_test.shape)
print("y_data matrix shape", y_data.shape)
print("y_test matrix shape", y_test.shape)
print("y_data: ",y_data)
```

```
Training matrix shape (50000, 32, 32, 3)
Testing matrix shape (10000, 32, 32, 3)
y_data matrix shape (50000, 1)
y_test matrix shape (10000, 1)
y_data:  [[6]
 [9]
 [9]
 ...
 [9]
 [1]
 [1]]
```

Let's first see the distribution of pixels in each image

```
## Step 7.2: Plot the histogram for the pixels in each image
from matplotlib import pyplot as plt
plt.hist(X_data[0].flatten(),)
plt.title("Distribution of Pixel values in training example")
plt.show()
```



Task: Load the above codes to your notebook, and make sure you can generate figures correctly.

Step 8: Prepare features of inputs for neural network training

First, the neural network will take a **single vector** for each training example, so we need to **reshape** the input so that each 32x32x3 image becomes a single 3072-dimensional vector.

Therefore, the whole dataset for neural network training should have input shape (N,D), where N is the total number of images, D is the total number of features

```
N_train = X_data.shape[0]  # the first dimension of the tensor is number of total images
D_train = 32*32*3   # the remaining dimensions of the tensor is the shape of image, you can also use X_data.shape[1]*X_data.shape[2]*X_data.shape[3]
X_data_flatten = X_data.reshape(N_train, D_train)
X_data_flatten = X_data_flatten.astype('float32')

N_test = X_test.shape[0]
D_test = 32*32*3
X_test_flatten = X_test.reshape(N_test,D_test)
X_test_flatten = X_test_flatten.astype('float32')

print("Training matrix shape", X_data_flatten.shape)
print("Testing matrix shape", X_test_flatten.shape)
```

```
Training matrix shape (50000, 3072)
Testing matrix shape (10000, 3072)
```

We'll also **scale** the inputs to be in the range [0-1] rather than [0-255]. We can use min-max normalization here:

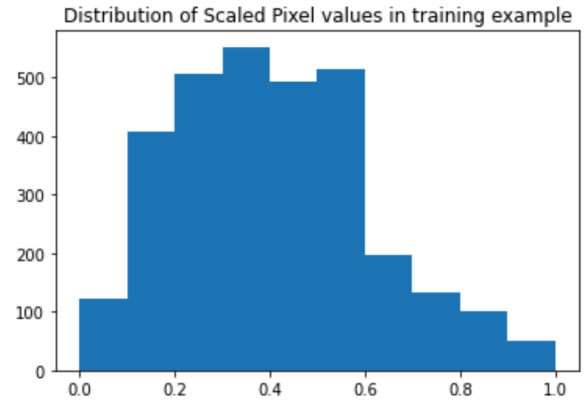
```
X_data_flatten /= 255
X_test_flatten /= 255
```

Let's visualize the histogram of the scaled data again,

```
from matplotlib import pyplot as plt
plt.hist(X_data_flatten[0].flatten(),)
plt.title("Distribution of Scaled Pixel values in training example")
plt.show()
```

Task: Check you notice any difference between the scaled image and original image?

A sample image is attached:



Question 17 4 pts

Step 9: Processing labels of inputs for the neural network

Our label information is saved in the variable 'y_train', please print the y_train using **print(y_data)**

```
print(y_data)
```

```
[[6]
 [9]
 [9]
 ...
 [9]
 [1]
 [1]]
```

For multi-class classification using the softmax function, we must modify the target matrices to be in the one-hot format, i.e.

'airplane' -> 0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

'automobile' -> 1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

'bird' -> 2 -> [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

etc.

Feature encoding for categorical values

❑ Variable Preprocessing (One-Hot encoder)

- **Convert textual/categorical column into numeric numbers**

✓ **One-hot Encoder.** Each categorical nominal value is uniquely represented by a k-dim vector of 0/1, k is the number of categories in particular column

- Create a new dummy variable for each category (sort categories by lexicographically order)
- Label the dummy variable as 1 if example falls in the category, otherwise, label the dummy variable as 0 i.e., given three classes, 'Setosa' -> [1, 0, 0], 'Versicolor' -> [0, 1, 0] , 'Virginica' -> [0, 0, 1]

ID	Label
1	Setosa
2	Setosa
3	Versicolor
4	Versicolor
5	Virginica
6	Virginica

Apply One-hot Encoder

- Expand one column leads to K columns

One-hot encoder label

ID	Class 0	Class 1	Class 2
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	0	0	1
6	0	0	1

We can apply tf.keras.utils.to_categorical() function from tensorflow to convert categorical features into one-hot encoding

```
labels_map = ['airplane', 'automobile', 'bird', 'cat', 'deer',
'dog', 'frog', 'horse', 'ship', 'truck']
nb_classes = len(labels_map)

import tensorflow as tf
y_data_categorical = tf.keras.utils.to_categorical(y_data, nb_cl
asses)
y_test_categorical = tf.keras.utils.to_categorical(y_test, nb_cl
asses)

print("y_data matrix shape", y_data_categorical.shape)
print("y_test matrix shape", y_test_categorical.shape)
print("y_test_categorical: ",y_test_categorical)
```

```
y_data matrix shape (50000, 10)
y_test matrix shape (10000, 10)
y_test_categorical:  [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]]
```

Task: Copy/paste your outputs to this box, and describe the changes in the shape of label matrix

Question 18 4 pts

Step 10: Data preparation is done, let's build/Train the neural network (make sure you are using GPU node in Colab)

Build the neural-network. Here we'll do a simple 3 layer fully connected network.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

## Set up training and validation dataset
X_train, X_val, y_train, y_val = train_test_split(X_data_flatten,y_data_cat
egorical, test_size=0.2, random_state=42)
def build_model(n_layers = 3, n_neurons = 1000):
   model = Sequential() # create Sequential model
   for i in range(n_layers-1):
       model.add(Dense(n_neurons, activation = 'relu')) # you can also try
other types of activation functions
   model.add(Dense(10, activation = 'softmax'))  # the output must be softm
ax for multi-class classifcation

   return model

model = build_model(n_layers = 3, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
train_history = model.fit(X_train,y_train, validation_data=(X_val,y_val), b
atch_size=128, epochs = 20)
```

```
Epoch 1/20
313/313 [==============================] - 7s 11ms/step - loss: 1.9560 - accuracy: 0.3047 - val_loss: 1.7857
Epoch 2/20
313/313 [==============================] - 3s 9ms/step - loss: 1.7037 - accuracy: 0.3903 - val_loss: 1.7717 -
Epoch 3/20
313/313 [==============================] - 3s 10ms/step - loss: 1.6166 - accuracy: 0.4210 - val_loss: 1.5909
Epoch 4/20
313/313 [==============================] - 3s 10ms/step - loss: 1.5496 - accuracy: 0.4455 - val_loss: 1.5773
Epoch 5/20
313/313 [==============================] - 3s 9ms/step - loss: 1.5019 - accuracy: 0.4647 - val_loss: 1.5626 -
Epoch 6/20
313/313 [==============================] - 3s 10ms/step - loss: 1.4585 - accuracy: 0.4809 - val_loss: 1.5163
Epoch 7/20
313/313 [==============================] - 3s 10ms/step - loss: 1.4326 - accuracy: 0.4895 - val_loss: 1.5045
Epoch 8/20
313/313 [==============================] - 3s 10ms/step - loss: 1.3990 - accuracy: 0.5009 - val_loss: 1.5120
```

Task: Finish your training, summarize what you have observed from the changes of loss, accuracy on training and validation. For instance, is loss of train data always decreasing? How about the loss and accuracy of validation set?

Question 19 4 pts

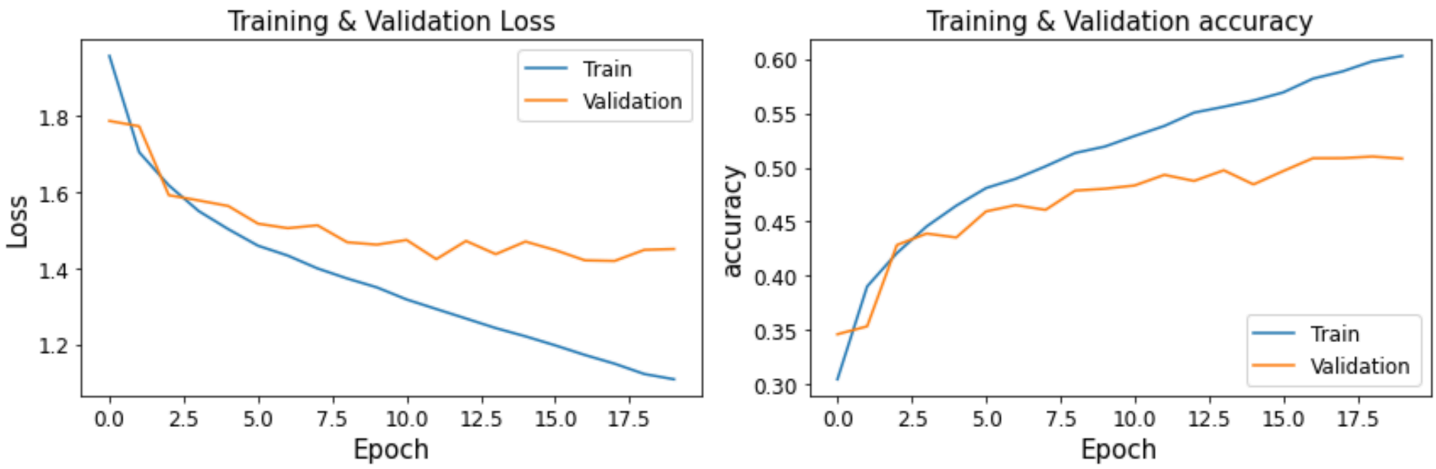
Step 11: Access the model training history & Plot the learning curves for training/validation

```
# Step 11.1:  Access the model training history
print(train_history.history.keys())
print(train_history.history['loss'])
```

```
# Step 11.2:  Plot the learning curves for training/validation
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
# Plot training & validation loss values
plt.plot(train_history.history['loss'], label='Train')
plt.plot(train_history.history['val_loss'], label='Validation')
plt.title('Training & Validation Loss', fontsize=15)
plt.ylabel('Loss', fontsize=15)
plt.xlabel('Epoch', fontsize=15)
plt.xticks( fontsize=12)
plt.yticks( fontsize=12)
plt.legend(loc='upper right', fontsize=12)
```

```
plt.subplot(1,2,2)
# Plot training & validation accuracy values
plt.plot(train_history.history['accuracy'], label='Train')
plt.plot(train_history.history['val_accuracy'], label='Validation')
plt.title('Training & Validation accuracy', fontsize=15)
```

```
plt.ylabel('accuracy', fontsize=15)

plt.xlabel('Epoch', fontsize=15)
plt.xticks( fontsize=12)
plt.yticks( fontsize=12)
plt.legend(loc='lower right', fontsize=12)
plt.tight_layout()
plt.show()
```



Task 1: Upload your learning curves to this box.

Note: There are several factors we can consider from the results above. E.x., can the accuracy continue to be improved? should we increase the epochs?

Question 20 6 pts

Step 12: Let's check the predicted labels of training data using the trained model

```
# Step 12:  Let's check the predicted labels of training data using the tra
ined model
train_predicted_labels = model.predict(X_train[0:5,:]) # here we only predi
ct the labels of first 5 images
print("Shape: ",train_predicted_labels.shape)
print(train_predicted_labels)
```

```
Shape:  (5, 10)
[[7.1642394e-03 2.0417511e-02 5.8358319e-02 1.2314646e-01 1.2265023e-01
  6.1892945e-02 5.2229351e-01 3.0718766e-02 1.0891345e-02 4.2466652e-02]
 [1.8542539e-02 3.7797371e-01 1.5277362e-01 2.9494673e-01 2.4560620e-03
  3.3927560e-02 2.4941329e-02 2.2092441e-02 1.4352743e-02 5.7993378e-02]
 [3.4238864e-02 1.0081425e-03 1.9210872e-01 5.7798613e-02 8.0612293e-03
  5.0500232e-01 1.8779364e-04 2.0064025e-01 4.2965428e-05 9.1108400e-04]
 [3.4422935e-03 1.9993747e-03 4.0256735e-02 1.7459357e-01 2.1026628e-02
  1.4515981e-01 5.8470178e-01 2.6251603e-02 3.9908002e-04 2.1690861e-03]
 [4.7785218e-04 2.4071908e-04 9.8804042e-02 5.8460778e-01 5.7127263e-02
  2.1520558e-01 1.0874617e-02 2.7854551e-02 4.1491996e-05 4.7660908e-03]]
None
```

Task 1: in the above matrix, each row represents one image, and has 10 values. What do those float numbers in each row represent?

Task 2: In the above matrix, how should we identify the predicted class for each image sample (row)?

Task 3: If we run the following code:

```
import numpy as np
np.argmax(train_predicted_labels,axis=1) # find the index of col
umn which has maximum value in each row
```

You will get similar output such as

array([6, 1, 5, 6, 3])

Explain how each integer in this array is derived and why they can be treated as the predicted class for each instance in each row (in above probability image)

Question 21 4 pts

Step 13: Evaluate the classification performance

```
# Step 13:  Evaluate the classification performance
from sklearn.metrics import accuracy_score
```

```
def evaluate_model(model,train_data,val_data,test_data):
   X_train,y_train = train_data
   X_val,y_val = val_data
   X_test,y_test = test_data
   # (1) make a prediction on training set to get probablities for all clas
ses, select the class that has maximum probablity
   y_train_pred = np.argmax(model.predict(X_train), axis=-1)
   # (2) calculate the training classification error
   Train_error_s = 1 - accuracy_score(np.argmax(y_train,axis=1), y_train_pr
ed)
   # (3) make a prediction on validation set
   y_val_pred = np.argmax(model.predict(X_val), axis=-1)
   # (4) calculate the validation classification error
   Val_error_s = 1 - accuracy_score(np.argmax(y_val,axis=1), y_val_pred)
   # (5) make a prediction on test set
   y_test_pred = np.argmax(model.predict(X_test), axis=-1)
   # (6) calculate the test classification error
   Test_error_s = 1 - accuracy_score(y_test, y_test_pred)
```

```
    # (7) reporting results
    print("Train error: ", Train_error_s)
    print("Validation error: ", Val_error_s)
    print("Test error: ", Test_error_s)
    return Train_error_s,Val_error_s,Test_error_s
```

```
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train,y_tr
ain),(X_val,y_val),(X_test_flatten,y_test))
```

```
Train error:  0.3606000000000003
Validation error:  0.4919
Test error:  0.5008
```

Task: Copy/Paste your training results to this box

Step 14: In the following steps, we will explore different techniques to improve model learning.

Let's record the improvement process for this practicum

```
# Step 14: Implement a function for visualizing the improvements over attem
pts
def visualize_improvement(improvement_log_train,improvement_log_val,improve
ment_log_test):
    import matplotlib.pyplot as plt
    plt.figure(figsize=(12,4))

    # Plot training error values
    plt.subplot(1,3,1)
    plt.plot(improvement_log_train, label='Train')
    plt.title('Training error', fontsize=15)
    plt.ylabel('Error', fontsize=15)
    plt.xlabel('Process', fontsize=15)
    plt.xticks(range(len(improvement_log_train)), fontsize=12)
    plt.yticks( fontsize=12)
    plt.legend(loc='upper right', fontsize=12)

    # Plot Validation error values
    plt.subplot(1,3,2)
    plt.plot(improvement_log_val, label='Validation')
    plt.title('Validation error', fontsize=15)
    plt.ylabel('Error', fontsize=15)
    plt.xlabel('Process', fontsize=15)
    plt.xticks(range(len(improvement_log_val)), fontsize=12)
    plt.yticks( fontsize=12)
    plt.legend(loc='upper right', fontsize=12)

    # Plot testing error values
    plt.subplot(1,3,3)
    plt.plot(improvement_log_test, label='Test')
    plt.title('Test error', fontsize=15)
    plt.ylabel('Error', fontsize=15)
    plt.xlabel('Process', fontsize=15)
    plt.xticks(range(len(improvement_log_test)), fontsize=12)
    plt.yticks( fontsize=12)

    plt.legend(loc='upper right', fontsize=12)
    plt.tight_layout()
    plt.show()
```

Here we record our first training performance and do visualization

```
improvement_log_train = []
improvement_log_val = []
improvement_log_test = []
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
```

```
improvement_log_test.append(Test_error_s)

visualize_improvement(improvement_log_train,improvement_log_val,improvement_log_test)
```

Task: Make sure you run this section's code correclty. Otherwise, you will not get correct results in the following step

Question 22 4 pts

Step 15: Save models to disk. We can save the latest model to the local disk, and avoid re-training in the future (this is also called a pre-trained model).

```
from tensorflow.keras.models import load_model
# Step 15.1: save model to local file
model.save("CIFAR10_model_simple.h5")

# Step 15.2: reload model from the local file
model_loaded = load_model("CIFAR10_model_simple.h5")

# Step 15.3: you are supposed to see same performance as previous one
y_test_pred = np.argmax(model.predict(X_test_flatten), axis=-1)
Test_error_s = 1 - accuracy_score(y_test, y_test_pred)

print("Test results: ",Test_error_s)
```

Task: Copy/Paste your results here. Are you getting same or similar classification error you saw in Step 13.

Question 23 4 pts

Step 16: second attempt: Check if feature normalization improves results

Feature scaling

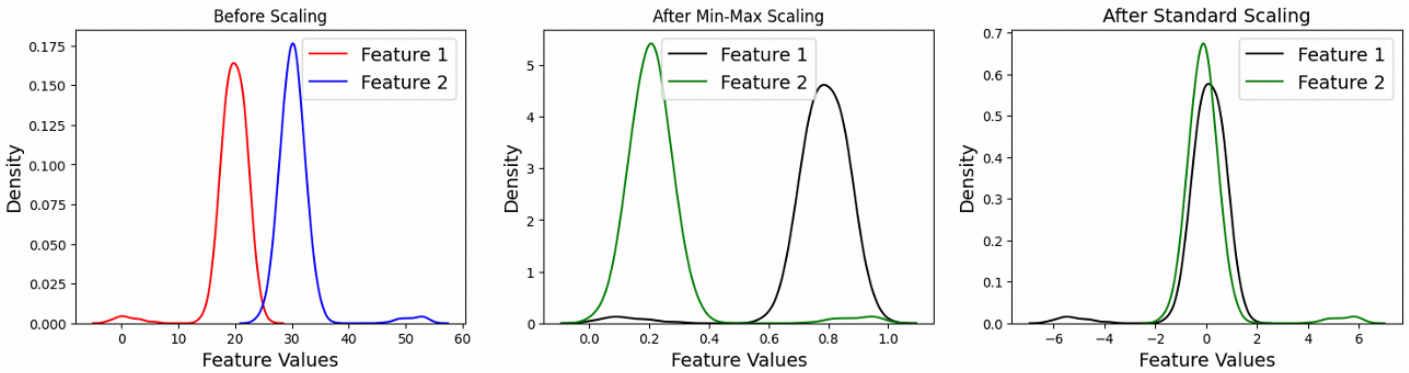
Min-Max normalization

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$$

Replace x_i with $\frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$ to make features have value between 0 and 1.

Mean normalization/Standardization

Replace x_i with $\frac{x_i - u_i}{\text{std}(x_i)}$ to make features have value with 0 mean and variance equals to 1.




```
# Step 16: second attempt: Check if feature normalization improves results
def build_model(n_layers = 2, n_neurons = 1000):
    model = Sequential() # create Sequential model
    for i in range(n_layers-1):
        model.add(Dense(n_neurons, activation = 'relu')) # you can also t
```

```
ry other types of activation functions

    model.add(Dense(10, activation = 'softmax'))  # the output must be soft
max for multi-class classifcation
    return model
```

```
model = build_model(n_layers = 3, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
```

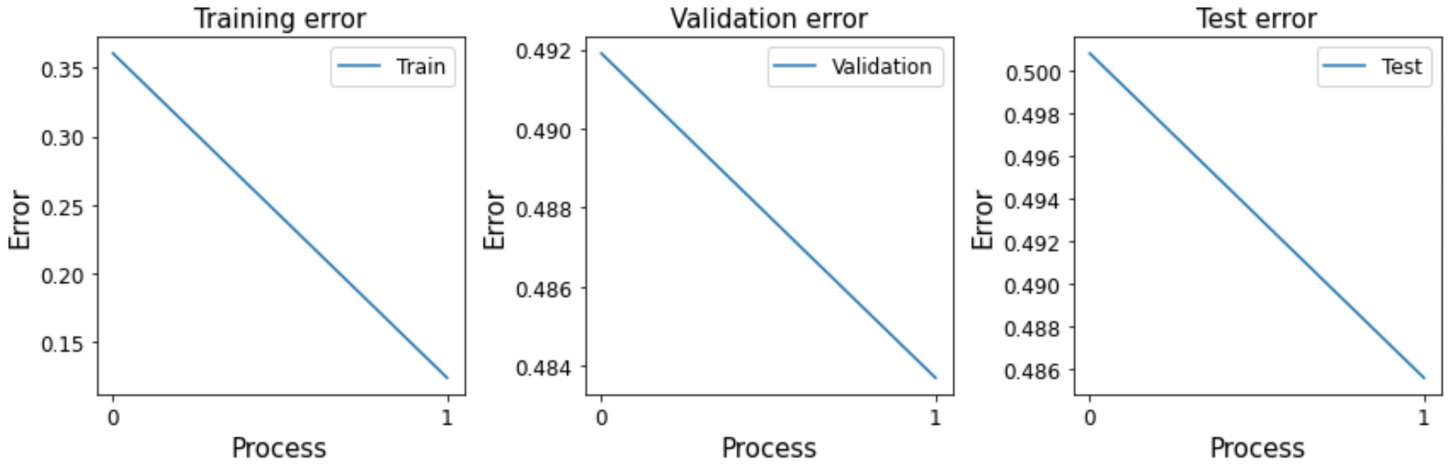
```
# Step 16.2: Apply feature normalization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.astype(np.float32))
X_val_s = scaler.transform(X_val.astype(np.float32))
X_test_s = scaler.transform(X_test_flatten.astype(np.float32))
```

```
# Step 16.3: Training the model on the scaled data
train_history = model.fit(X_train_s,y_train, validation_data=(X_val_s,y_va
l), batch_size=128, epochs = 20) # Make sure using scaled training and scal
ed validation.

# Step 16.4: Evaluate this model again to see any improvements
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train),(X_val_s,y_val),(X_test_s,y_test))
```

```
# Step 16.5: Let's visualize the improvements
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)
```

```
visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.12372499999999997
Validation error:  0.4837
Test error:  0.48560000000000003
```



Task: Upload your image of results to this box or show this image in your notebook.

Question 24 4 pts

Step 17: Third attempt: increase model complexity by adding more hidden layers

```
# Step 17: third attempt: increase model complexity by adding more hidden l
ayers
def build_model(n_layers = 5, n_neurons = 1000):
    model = Sequential() # create Sequential model
    for i in range(n_layers-1):
          model.add(Dense(n_neurons, activation = 'relu')) # you can also t
```

```
ry other types of activation functions

    model.add(Dense(10, activation = 'softmax'))  # the output must be soft
max for multi-class classifcation
    return model
```

```
# Step 17.1: let's increase the number of layers in model definition
model = build_model(n_layers = 5, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
```

```
# Step 17.2: let's re-use Step 16's code, and include the normalization ste
p due to its improvement.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.astype(np.float32))
X_val_s = scaler.transform(X_val.astype(np.float32))
X_test_s = scaler.transform(X_test_flatten.astype(np.float32))

# Step 17.3: Let's start retraining the model
train_history = model.fit(X_train_s,y_train, validation_data=(X_val_s,y_va
l), batch_size=128, epochs = 20)
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train),(X_val_s,y_val),(X_test_s,y_test))
```

```
313/313 [==============================] - 4s 12ms/step - loss: 1.4949 - accuracy: 0.4682 - val_loss: 1.5167 - val_accuracy:
Epoch 3/20
313/313 [==============================] - 4s 13ms/step - loss: 1.3800 - accuracy: 0.5138 - val_loss: 1.4976 - val_accuracy:
Epoch 4/20
313/313 [==============================] - 4s 13ms/step - loss: 1.2810 - accuracy: 0.5441 - val_loss: 1.4555 - val_accuracy:
Epoch 5/20
313/313 [==============================] - 4s 12ms/step - loss: 1.1983 - accuracy: 0.5723 - val_loss: 1.4463 - val_accuracy:
Epoch 6/20
313/313 [==============================] - 4s 12ms/step - loss: 1.1210 - accuracy: 0.5980 - val_loss: 1.4286 - val_accuracy:
Epoch 7/20
313/313 [==============================] - 4s 12ms/step - loss: 1.0389 - accuracy: 0.6285 - val_loss: 1.4503 - val_accuracy:
Epoch 8/20
313/313 [==============================] - 4s 13ms/step - loss: 0.9786 - accuracy: 0.6487 - val_loss: 1.5087 - val_accuracy:
Epoch 9/20
313/313 [==============================] - 4s 12ms/step - loss: 0.9000 - accuracy: 0.6767 - val_loss: 1.5355 - val_accuracy:
Epoch 10/20
313/313 [==============================] - 4s 12ms/step - loss: 0.8193 - accuracy: 0.7005 - val_loss: 1.5996 - val_accuracy:
Epoch 11/20
313/313 [==============================] - 4s 12ms/step - loss: 0.7616 - accuracy: 0.7265 - val_loss: 1.7651 - val_accuracy:
Epoch 12/20
313/313 [==============================] - 4s 13ms/step - loss: 0.6951 - accuracy: 0.7489 - val_loss: 1.7617 - val_accuracy:
```

```
# Step 17.4: Let's visualize the improvements
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)
```

```
visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.10504999999999998
Validation error:  0.4797
Test error:  0.48960000000000004
```

Task 1: Upload your image of results to this box or show this image in your notebook. What's your conclusion in this step?

Task 2: In the above training process, did you observe issues during the training? For instance, the training loss is decreasing, but validation loss is increasing. This is overfitting! We should take care of it. Did you also observe similar patterns.

Question 25 4 pts

Step18 In Step 17, it seems overfitting appears, you will see overfitting by increasing epoches.

```
# Step 18: forth attempt: increase the epoches during training
def build_model(n_layers = 5, n_neurons = 1000):
    model = Sequential() # create Sequential model
    for i in range(n_layers-1):
          model.add(Dense(n_neurons, activation = 'relu')) # you can also t
ry other types of activation functions

    model.add(Dense(10, activation = 'softmax'))  # the output must be soft
max for multi-class classifcation
    return model
```

```
# Step 18.1: let's use the same architecture in Step 17
model = build_model(n_layers = 5, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
```

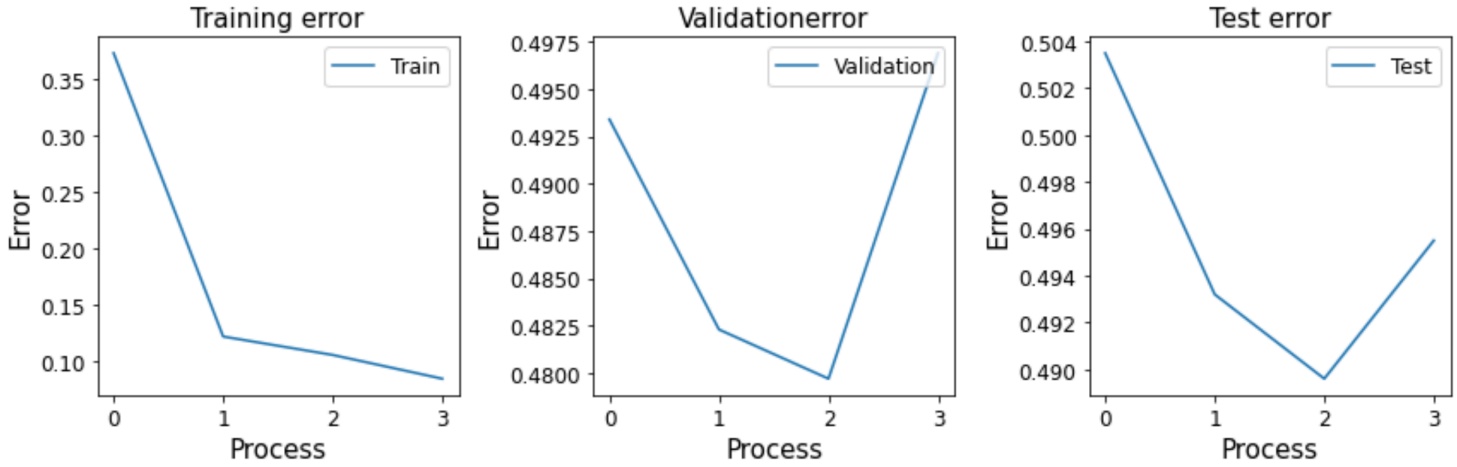
```
# Step 18.2: let's re-use Step 16's code, and include the normalization ste
p due to its improvement.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.astype(np.float32))
X_val_s = scaler.transform(X_val.astype(np.float32))
X_test_s = scaler.transform(X_test_flatten.astype(np.float32))

# Step 18.3: Let's start retraining the model with epochs = 30
train_history = model.fit(X_train_s,y_train, validation_data=(X_val_s,y_va
l), batch_size=128, epochs = 30)
```

```
# Step 18.4: let's evaluate the model again, I expected to see worse valida
tion results due to overfitting
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train),(X_val_s,y_val),(X_test_s,y_test))
```

```
# Step 18.5: Let's visualize the improvements
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)
```

```
visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.08389999999999997
Validation error:  0.4969
Test error:  0.49550000000000005
```



Task 1: Did you observe a similar overfitting pattern? If not, increase epochs to 40 or 50. Sometimes the validation classification accuracy may not improve when overfitting happens.

Task 2: Upload your image of performance results to this box. What's your conclusion in this step?

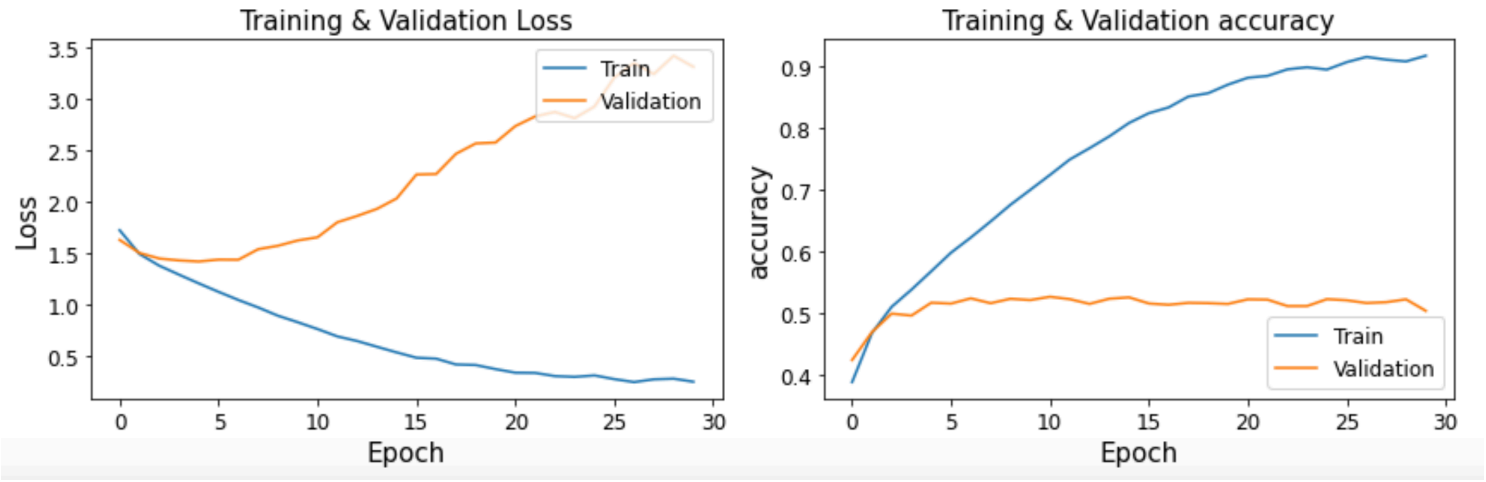
Task 3: List at least two solutions to avoid overfitting

Question 26 4 pts

Step 19: In previous step, as you see, the overfitting indeed happens. Looking into accuracy may not be very straightforward. If we visualize the learning curves from

the training step, you will get plots as follows:

```
# Step 19: Visualize the learning curves during training to diagnose the ov
erfitting
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)

# Step 19.1: Plot training & validation loss values
plt.plot(train_history.history['loss'], label='Train')
plt.plot(train_history.history['val_loss'], label='Validation')
plt.title('Training & Validation Loss', fontsize=15)
plt.ylabel('Loss', fontsize=15)
plt.xlabel('Epoch', fontsize=15)
plt.xticks( fontsize=12)
plt.yticks( fontsize=12)
plt.legend(loc='upper right', fontsize=12)

plt.subplot(1,2,2)

# Step 19.2: Plot training & validation accuracy values
plt.plot(train_history.history['accuracy'], label='Train')
plt.plot(train_history.history['val_accuracy'], label='Validation')
plt.title('Training & Validation accuracy', fontsize=15)
plt.ylabel('accuracy', fontsize=15)
plt.xlabel('Epoch', fontsize=15)
plt.xticks( fontsize=12)
plt.yticks( fontsize=12)
plt.legend(loc='lower right', fontsize=12)
plt.tight_layout()
plt.show()
```



Task: Describe what you have observed from the above learning curves. What are issues we are facing?

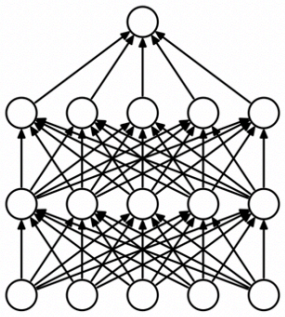
Question 27 4 pts

Step 20: In this case, let's add the dropout regularization layer to reduce overfitting

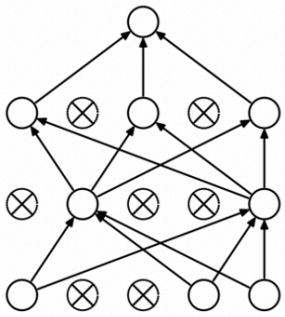
Tips and Tricks in Neural Network - Overfitting

❖ Preventing Overfitting by Dropout

- Remove randomly selected units according to a drop-out rate (0.5).
- Ignoring selected neurons by removing them during the training phase.
- Individual nodes are either removed with probability 1-p or kept with probability p
- Technically, we can sample an array of independent Bernoulli Distribution (u1 = np.random.binomial(1, p, size=n)), i.e., np.array([1, 0, 0, 1, 1, 0, 1, 0]) for 8 neurons
- Equivalent to training many networks with different number of hidden nodes and different connections.



(a) Standard Neural Net



(b) After applying dropout.

Tips and Tricks in Neural Network - Overfitting

On MNIST dataset

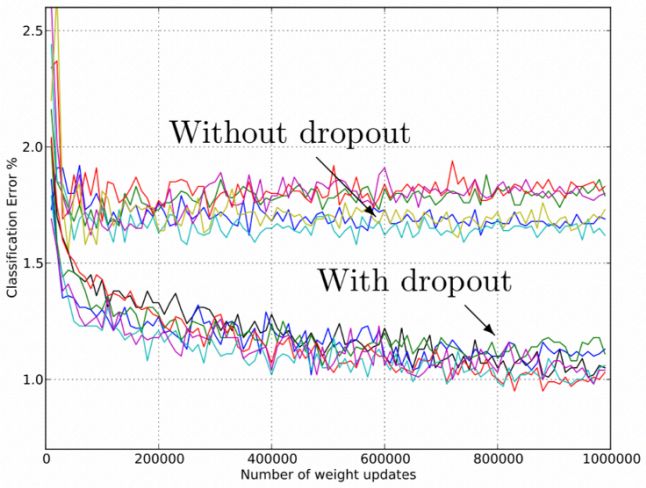


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

- **Figure** indicates dropout network will consistently perform better than without Dropout in the test dataset.
- **Dropout** network perform worse on the train dataset
- **High accuracy** on training data, but **low accuracy** on test, you need consider adding dropout layer to avoid overfitting.

Tips and Tricks in Neural Network - Overfitting

❖ Preventing Overfitting by Dropout in Keras

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

```
model = Sequential() # create Sequential model
model.add(Dropout(0.2, input_shape=(22,)))
model.add(Dense(100, activation = 'relu')) # input layer -> hidden layer 1
model.add(Dropout(0.2))
model.add(Dense(100, activation = 'relu')) # input layer -> hidden layer 1
model.add(Dropout(0.2))
model.add(Dense(1, activation = 'sigmoid')) # hidden layer 2 -> output layer
model.compile(loss = "binary_crossentropy", optimizer = "SGD", metrics = ['accuracy'])
model.fit(X_train,y_train, validation_data=(X_val,y_val), epochs = 100)
```

Dropout can be added to input layer

Dropout can be added to hidden layer

```
## Step 20: add a dropout layer to avoid overfitting
from tensorflow.keras.layers import Dropout
def build_model(n_layers = 2, n_neurons = 1000):
    model = Sequential() # create Sequential model
    for i in range(n_layers-1):
        model.add(Dense(n_neurons, activation = 'relu'))
        model.add(Dropout(0.2))
```

```
    model.add(Dense(10, activation = 'softmax'))
    return model
```

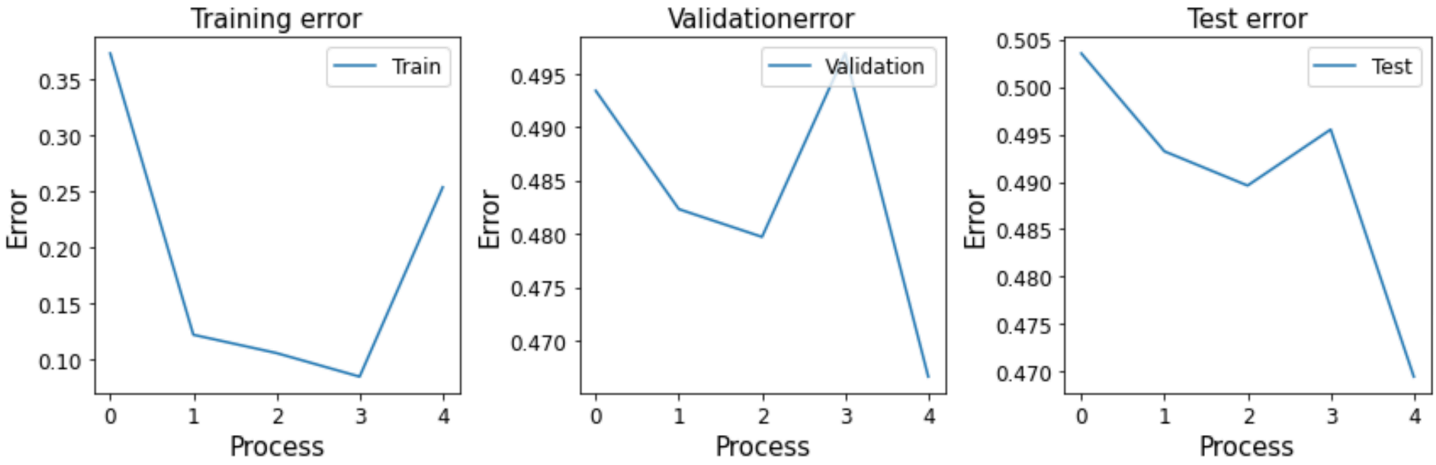
```
# Step 20.1: let's use the same architecture in Step 17
model = build_model(n_layers = 5, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
```

```
# Step 20.2: let's re-use Step 16's code, and include the normalization ste
p due to its improvement.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.astype(np.float32))
X_val_s = scaler.transform(X_val.astype(np.float32))
X_test_s = scaler.transform(X_test_flatten.astype(np.float32))

# Step 20.3: Let's start retraining the model with epochs = 30
train_history = model.fit(X_train_s,y_train, validation_data=(X_val_s,y_va
l), batch_size=128, epochs = 30)
```

```
# Step 20.4: let's evaluate the model again, I expected to see better valid
ation results as we use regularizations
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train),(X_val_s,y_val),(X_test_s,y_test))
```

```
# Step 20.5: Let's visualize the improvements
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)

visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.25365000000000004
Validation error:  0.4666
Test error:  0.46950000000000003
```



Task 1: Upload your image of performance results. Did you get similar results as above image?

Task 2: Explain why training classification error increases after the dropout regularization is included?

Question 28 4 pts

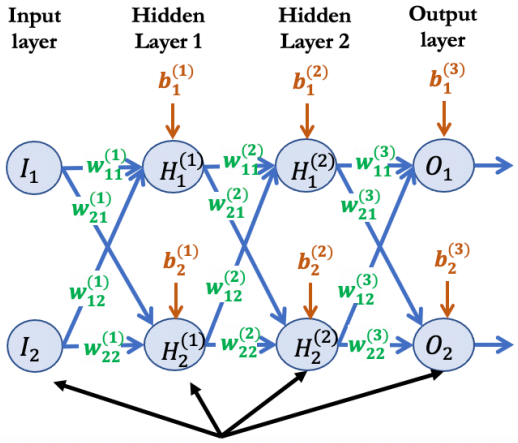
Step 21: In Step 20, as you see, adding dropout can reduce overfitting, and further improves the validation performance.

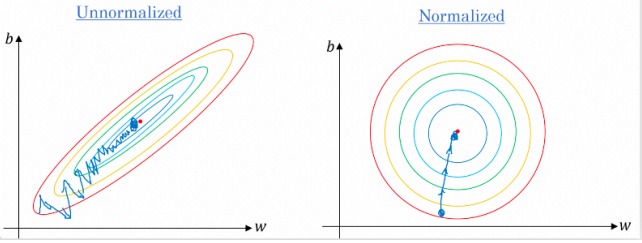
Now the training accuracy is still not high, how about adding the batch normalization layers

Tips and Tricks in Neural Network - Optimization

❖ Batch Normalization

- ➢Output values from each layer are inputs for next layer
- ➢Large output values from hidden nodes hinder learning using gradient descent.
- ➢Same as scaling input data, the output values from each hidden layer should also be normalized.
- ➢Speed up training





Tips and Tricks in Neural Network - Optimization

❖ Batch Normalization

➢ Normalizing the output of each node in each hidden layer

normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. For a layer with d-dimensional input $\mathrm{x} = (x^{(1)} \ldots x^{(d)})$, we will normalize each dimension

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

where the expectation and variance are computed over the training data set. As shown in (LeCun et al., 1998b), such normalization speeds up convergence, even when the features are not decorrelated.

Tips and Tricks in Neural Network - Optimization

❖ Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- Mean and the variance are fixed during prediction
- Gamma γ and Beta β are two trainable parameters to each layer

https://arxiv.org/pdf/1502.03167v3.pdf

Tips and Tricks in Neural Network - Optimization

❖ Batch Normalization

```
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import BatchNormalization

model = Sequential() # create Sequential model
model.add(Dense(100, activation = 'relu')) # input layer -> hidden layer 1
model.add(BatchNormalization())
model.add(Dense(100, activation = 'relu')) # input layer -> hidden layer 1
model.add(BatchNormalization())
model.add(Dense(1, activation = 'sigmoid')) # hidden layer 2 -> output layer

model.compile(loss = "binary_crossentropy", optimizer = "SGD", metrics = ['accuracy'])
model.fit(X_train,y_train, validation_data=(X_val,y_val), epochs = 100)
model.evaluate(data_test_features, data_test_label)
```

```
# Step 21.1: Adding normalization layer
from keras.layers import Dropout
from keras.layers import BatchNormalization
```

```
def build_model(n_layers = 2, n_neurons = 1000):
    model = Sequential() # create Sequential model
    for i in range(n_layers-1):
        model.add(Dense(n_neurons, activation = 'relu'))
        model.add(BatchNormalization()) # add normalization here
        model.add(Dropout(0.2))
    model.add(Dense(10, activation = 'softmax'))
    return model
```

```
# Step 21.2: retraining the model with same settings
model = build_model(n_layers = 5, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
```

```
# Step 21.2: let's re-use Step 16's code, and include the normalization ste
p due to its improvement.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.astype(np.float32))
X_val_s = scaler.transform(X_val.astype(np.float32))
X_test_s = scaler.transform(X_test_flatten.astype(np.float32))

# Step 21.3: Let's start retraining the model with epochs = 30
train_history = model.fit(X_train_s,y_train, validation_data=(X_val_s,y_va
l), batch_size=128, epochs = 30)
```

```
# Step 21.4: let's evaluate the model again
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train),(X_val_s,y_val),(X_test_s,y_test))
```

```
# Step 21.5: Let's visualize the improvements
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)

visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.046599999999999975
Validation error:  0.4315
Test error:  0.43400000000000005
```

Task 1: Upload your image of performance results. Did you get similar results as above image?

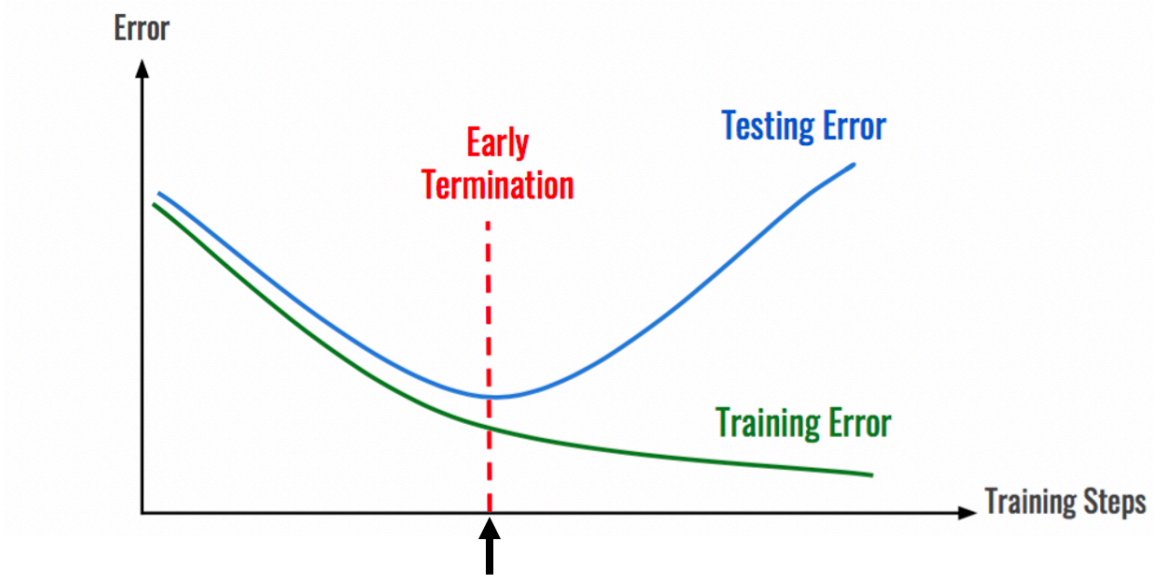
Task 2: What are your best classification error now for training, validation, and test.

Question 29 4 pts

Step 22: Another option to avoid overfitting is using early stop technique during training. We need to add early stop to avoid overfitting as well

Tips and Tricks in Neural Network - Overfitting

❖ Preventing Overfitting by Early Stopping



Early stopping: stopping the training process before the training algorithm passes that point.

Tips and Tricks in Neural Network - Overfitting

❖ Preventing Overfitting by Early Stopping in Keras

➢ Early Stopping:

Set patience count = 0
After each epoch:

1. Check whether the monitor 'metric' has improved with respect to the best value found so far
2. If it has not improved, it increases the count of 'times not improved (patience) by 1
3. If it did actually improve, it resets the patient count to 0.

➢ Model Checkpoints:

After each epoch:

1. Save the best performing model automatically
2. Saves the entire model or its weights to an HDF5 file

Tips and Tricks in Neural Network - Overfitting

❖ Preventing Overfitting by Early Stopping in Keras

➢ Define callbacks to monitor EarlyStopping

```
from keras.callbacks import EarlyStopping, ModelCheckpoint
keras_callbacks   = [
      EarlyStopping(monitor='val_accuracy', patience=5, mode='min', min_delta=0.0001),
      ModelCheckpoint('./checkmodel.h5', monitor='val_accuracy', save_best_only=True, mode='min')
]
model = Sequential() # create Sequential model
model.add(Dense(100, activation = 'relu')) # input layer -> hidden layer 1
model.add(Dense(100, activation = 'relu')) # input layer -> hidden layer 1
model.add(Dense(1, activation = 'sigmoid')) # hidden layer 2 -> output layer
model.compile(loss = "binary_crossentropy", optimizer = "SGD", metrics = ['accuracy'])
model.fit(X_train,y_train, validation_data=(X_val,y_val), epochs = 100, callbacks=keras_callbacks)
```

```
Epoch 1/100
2/2 [==============================] - 0s 212ms/step - loss: 0.7041 - accuracy: 0.4286 - val_loss: 0.7062 - val_accuracy: 0.5000
Epoch 2/100
2/2 [==============================] - 0s 8ms/step - loss: 0.7013 - accuracy: 0.5714 - val_loss: 0.7034 - val_accuracy: 0.5417
Epoch 3/100
2/2 [==============================] - 0s 7ms/step - loss: 0.6984 - accuracy: 0.5893 - val_loss: 0.7006 - val_accuracy: 0.5000
Epoch 4/100
2/2 [==============================] - 0s 7ms/step - loss: 0.6957 - accuracy: 0.5714 - val_loss: 0.6982 - val_accuracy: 0.5000
Epoch 5/100
2/2 [==============================] - 0s 7ms/step - loss: 0.6932 - accuracy: 0.6071 - val_loss: 0.6956 - val_accuracy: 0.5000
Epoch 6/100
2/2 [==============================] - 0s 7ms/step - loss: 0.6906 - accuracy: 0.6071 - val_loss: 0.6933 - val_accuracy: 0.5000
<tensorflow.python.keras.callbacks.History at 0x7fb3efc44978>
```

```
# Step 22.1: Add early stop into model training
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
keras_callbacks = [
      EarlyStopping(monitor='val_loss', patience=10, mode='min', min_delta=
0.0001),
      ModelCheckpoint('./checkmodel.h5', monitor='val_loss', save_best_only
=True, mode='min')
]
```

```
# Step 21.2: retraining the model with same settings
model = build_model(n_layers = 5, n_neurons = 1000)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metric
s = ['accuracy'])
```

```
# Step 21.2: let's re-use Step 16's code, and include the normalization ste
p due to its improvement.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train.astype(np.float32))
X_val_s = scaler.transform(X_val.astype(np.float32))
X_test_s = scaler.transform(X_test_flatten.astype(np.float32))
```

```
# Step 21.3: Let's start retraining the model with epochs = 30
train_history = model.fit(X_train_s,y_train, validation_data=(X_val_s,y_va
l), batch_size=128, epochs = 30, callbacks=keras_callbacks) # add callbacks
here
```

```
# Step 21.4: let's evaluate the model again
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train),(X_val_s,y_val),(X_test_s,y_test))
```

```
# Step 21.5: Let's visualize the improvements
improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)

visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.153725
Validation error:  0.4305
Test error:  0.43200000000000005
```

Task: If you check the training process, which epoch does the training stop at? For instance, in the below training process, the training stops at epoch 19 out of 30 because the validation loss is not improved in the last 10 epochs. So the training stops.

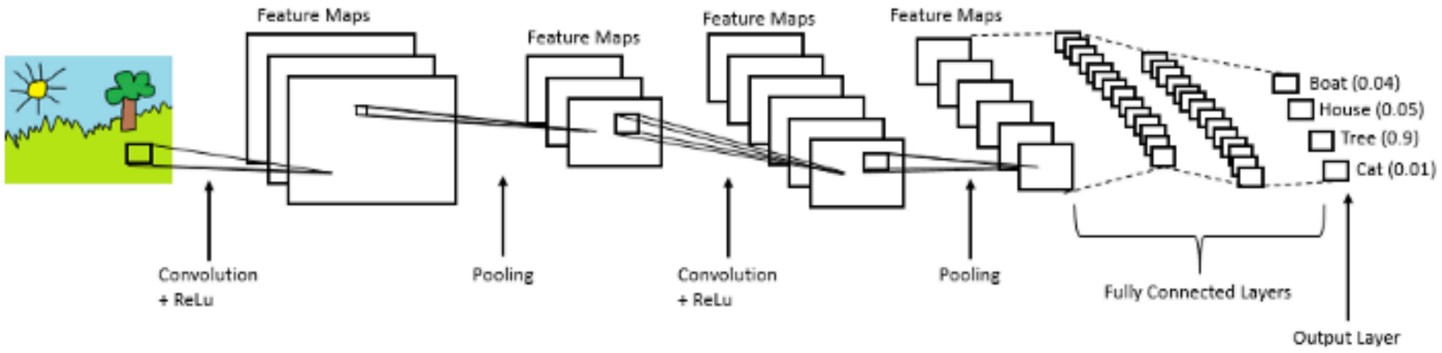
Describe your observation from your training process.

```
Epoch 9/30
313/313 [==============================] - 5s 17ms/step - loss: 1.1008 - accuracy: 0.6077 - val_loss: 1.3182 - val_accuracy:
Epoch 10/30
313/313 [==============================] - 5s 17ms/step - loss: 1.0606 - accuracy: 0.6211 - val_loss: 1.3270 - val_accuracy:
Epoch 11/30
313/313 [==============================] - 5s 16ms/step - loss: 1.0142 - accuracy: 0.6383 - val_loss: 1.3378 - val_accuracy:
Epoch 12/30
313/313 [==============================] - 5s 17ms/step - loss: 0.9649 - accuracy: 0.6530 - val_loss: 1.3515 - val_accuracy:
Epoch 13/30
313/313 [==============================] - 5s 16ms/step - loss: 0.9303 - accuracy: 0.6694 - val_loss: 1.3352 - val_accuracy:
Epoch 14/30
313/313 [==============================] - 5s 16ms/step - loss: 0.8845 - accuracy: 0.6846 - val_loss: 1.3507 - val_accuracy:
Epoch 15/30
313/313 [==============================] - 5s 17ms/step - loss: 0.8485 - accuracy: 0.6946 - val_loss: 1.3440 - val_accuracy:
Epoch 16/30
313/313 [==============================] - 5s 17ms/step - loss: 0.8000 - accuracy: 0.7155 - val_loss: 1.4358 - val_accuracy:
Epoch 17/30
313/313 [==============================] - 5s 17ms/step - loss: 0.7659 - accuracy: 0.7259 - val_loss: 1.4260 - val_accuracy:
Epoch 18/30
313/313 [==============================] - 5s 16ms/step - loss: 0.7151 - accuracy: 0.7418 - val_loss: 1.4358 - val_accuracy:
Epoch 19/30
313/313 [==============================] - 5s 16ms/step - loss: 0.6884 - accuracy: 0.7529 - val_loss: 1.4498 - val_accuracy:
Train error:  0.140675
Validation error:  0.4458
Test error:  0.44179999999999997
```

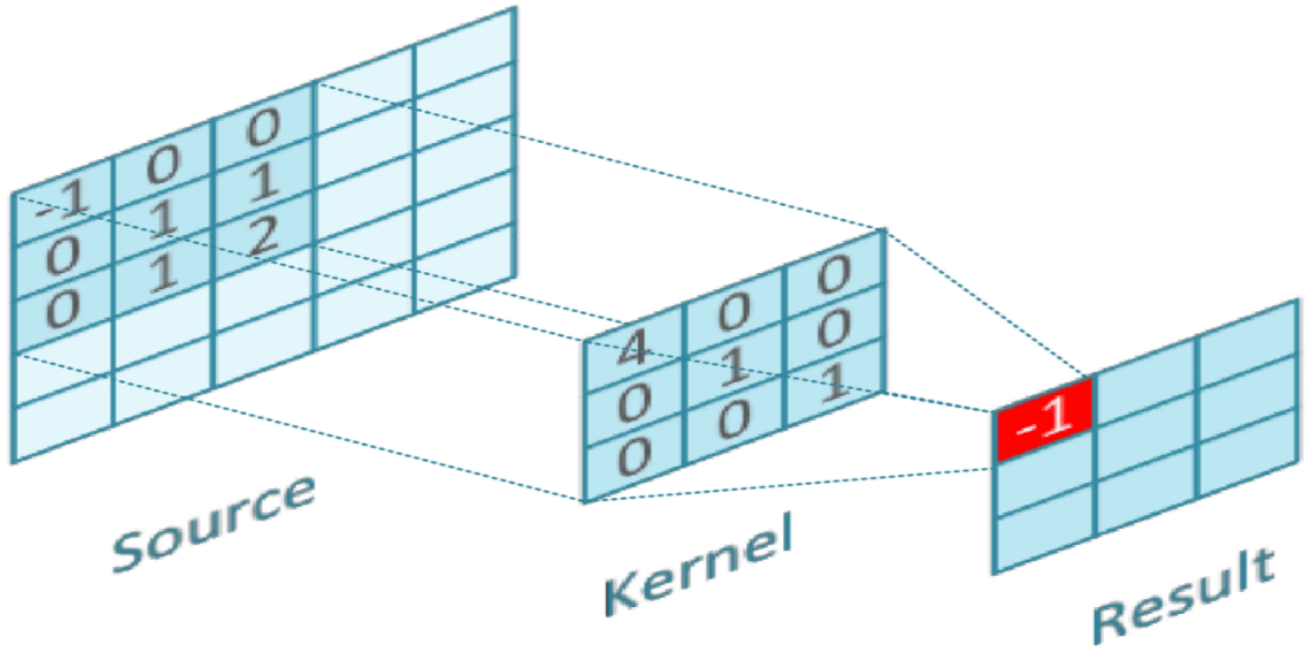
Question 30 0 pts

Step 23: Let's explore a more advanced neural network for image classification.

Introduction to Convolutional Neural Network



The convolution neural network is introduced in the next topic. Here, we need to know that 2D-CNN accepts a matrix as an input instead of a single-dimensional vector as the standard neural network does.



CNN will use a small-size matrix called 'kernel' (i.e., 3x3) to slide over all the image locations by following the directions of left to right and up to down. The kernel matrix will multiply the window within the input feature matrix in each location and then sum up values. The sum value will be treated as the convolution results and saved in the corresponding location in the new feature map'.

Now Let's simply replace the standard neural network with the convolutional neural network and evaluate how much improvement can be derived.

Question 31 4 pts

Step 23: Use convolutional neural network to improve the classification

```
# Step 23: Use a convolutional neural network to improve the classification
# Step 23.1: Load dataset
from keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
```

```
# Step 23.2: Process the labels to get one-hot encoding
num_classes = len(labels_map)
from keras.utils.np_utils import to_categorical
y_train_onehot = to_categorical(y_train, num_classes)
y_test_onehot = to_categorical(y_test, num_classes)
```

```
# Step 23.3: Normalize the features using min-max
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

```

```

```
# Step 23.4: Get training, validation dataset
X_train_s, X_val, y_train_s, y_val = train_test_split(X_train,y_train_oneho
```

```
t, test_size=0.2, random_state=42)
```

```
# Step 23.5: Define convolutional neural network
```

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from keras.layers import Conv2D,MaxPooling2D
from keras import regularizers
from keras.layers import Flatten,BatchNormalization,Dropout
```

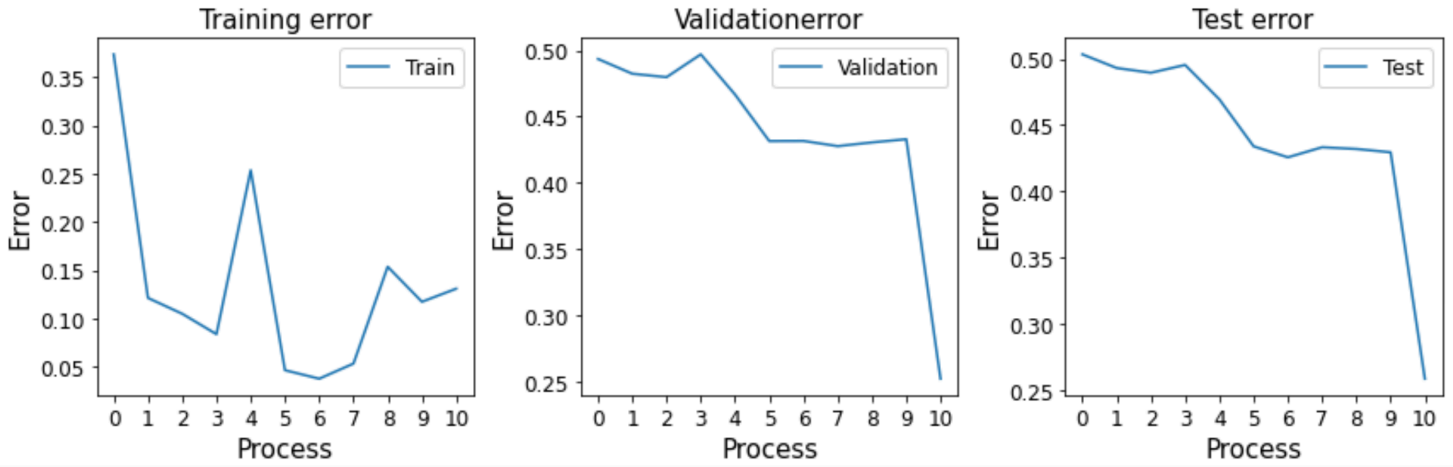
```
model = Sequential() # create Sequential model
model.add(Conv2D(32, (3,3), input_shape=(32,32,3), padding='same', activati
on = 'relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3,3), padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(10, activation = 'softmax'))

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

```
# Step 23.6: Start model training
history = model.fit(X_train_s,y_train_s, validation_data=(X_val,y_val), bat
ch_size=64, epochs = 20)
```

Step 23.7: Let's evaluate model and visualize the improvements again

```
Train_error_s,Val_error_s,Test_error_s = evaluate_model(model,(X_train_s,y_
train_s),(X_val,y_val),(X_test,y_test))

improvement_log_train.append(Train_error_s)
improvement_log_val.append(Val_error_s)
improvement_log_test.append(Test_error_s)
visualize_improvement(improvement_log_train,improvement_log_val,improvement
_log_test)
```

```
Train error:  0.13097499999999995
Validation error:  0.25229999999999997
Test error:  0.25839999999999996
```



Task: What's the best performance you get from Convolutional neural network. Upload the image of classification performance.

Question 32 4 pts

Step 24: In summary, we are able to reduce the test error from 0.503 (simple nerual network) to 0.258 (CNN)

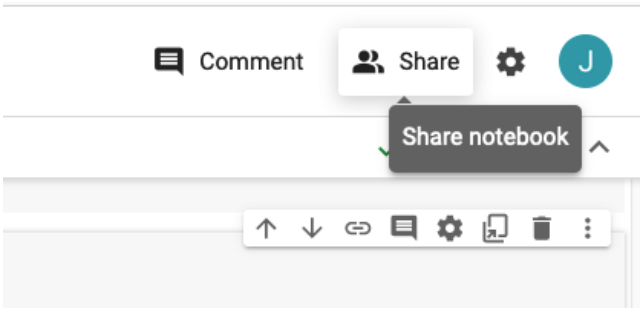
Task: summarize what you have learned from this practicum

Question 33 1 pts

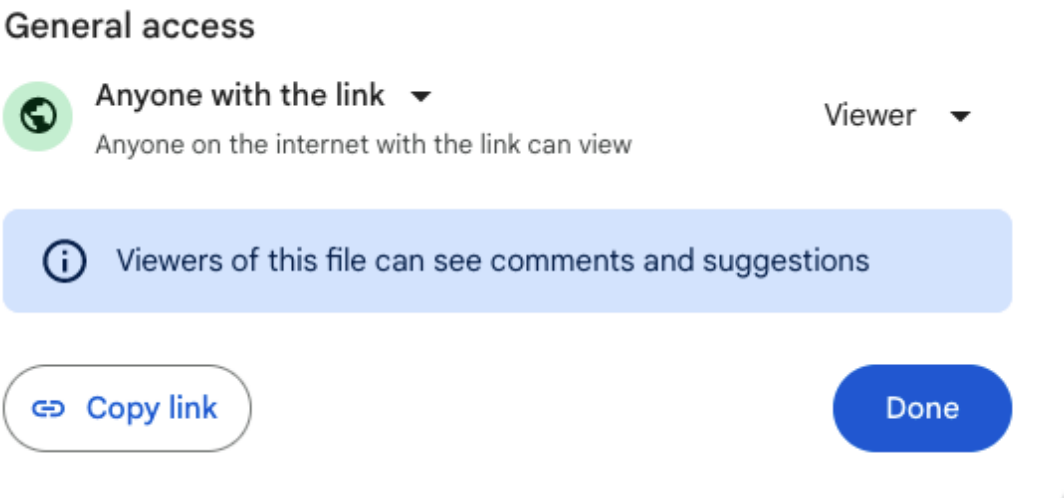
(30 points) Submission requirement I

Share your google colab link with us. See steps below

Step 1: Click 'Share' on top right corner of Google Colab



Step 2: Set 'General Access' to 'Anyone with the link', and click 'Copy Link'



Step 3: Copy & Paste your Google Colab link in this textbox

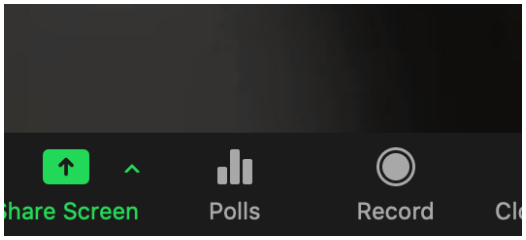
Question 34 1 pts

Submission requirement II

Requirement:

Part II: Every student needs to give 10-15 min (or longer) demo to describe how you complete the questions in Lab04. Everyone needs to have your own recording. The detailed presentation activities are attached below:

Step 1. Open zoom (myslu -> zoom), find the following buttons



Step 2. Click "share screen" button

Step 3. Click "record" button

Step 4. Present the following items:

a. (5 points) Give a demo to show **how the neural network model is built for multi-classification on iris data, and how hyper-parameters in neural network are applied.**

b. (5 points) Give a demo to show **how images can be processed in python.** You need explain your code structures and run the codes without errors.

c. (5 points) Give a demo to show **how different tips & tricks in neural network model can be applied to address overfitting & improve the performance.** You need explain your code structures and run the codes without errors.

c. (5 points) Summarize what you have learned in this lab.

Limit your presentation no longer than 15min. Please do not read codes during presentation. You should explain the purposes of codes in each step.

Step 5. Stop recording

Step 6. Submit the recording link as following format in the text box of submission page

Zoom recording link:XXXXXXXXX
Passcode: XXXXXX

(do not download recording video)

Upload

Choose a File

Not saved Submit Quiz