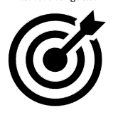
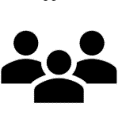
**LSPU Self-Paced Learning Module (SLM)**

|  |  |
| --- | --- |
| **Course** | Advance Machine Learning |
| **Sem/AY** | Second Semester/2020-2021 |
| **Module No.** | Module 2 |
| **Lesson Title** | **Deep Learning Algorithm for Images** |
| **Week Duration** | **Week 6** |
| **Date** | **April 12 – May 14** |
| **Description of the Lesson** | This topic focuses on the usage of deep learning algorithm in images its architecture and training process. The topic will discuss the most established deep learning algorithm used in images. Also, an introduction to computer vision is included in this module. |

****

**Learning Outcomes**

|  |  |
| --- | --- |
| **Intended Learning Outcomes** | Define and apply basic concepts and tools from optimization, especially geared towards solving machine learning problems. |
| **Targets/ Objectives** | At the end of the lesson, students should be able to:  1. Determine what is convexity, duality, and Lagrange multipliers.  2. Familiarize to optimization algorithms including gradient descent, stochastic gradient descent, Newton’s method, and Quasi-Newton methods.  3. Understand specialized algorithms tailored towards solving Linear Programming and Quadratic Programming problems which often arise in machine learning problems. |



**Student Learning Strategies**

|  |  |
| --- | --- |
| **Online Activities (Synchronous/**  **Asynchronous)** | 1. Online Discussion via Google Meet   For this module you will be directed to engage in a one-hour synchronous discussion and the rest will be asynchronous and offline activities. To access to the online course materials please check your Google Classroom Account.  The one-hour synchronous discussion will be on the schedule reflected on your certificate of registration and will be done in Google Meet. Please be reminded to prepare and be ready 15 minutes prior to the said schedule to lessen connection issues. For those who cannot attend the session recordings will be available after and will be posted with 24 hours. In case you may not be able to attend the session, ensure to notify your instructor. Please be reminded of the web conference etiquettes and reminders uploaded on you LMS.  You will be given time to complete all performance tasks and activity provided on the LMS as listed below:  1. Watch the video lecture  2. Read the SLM  3. Accomplish performance tasks using work sheet provided available in Google Classroom and submit at the submission link provided  4. For further study- Watch this video with this url: How to Develop a Good Research Topic  https://youtu.be/nXNztCLYgxc  5. Attend the synchronous class - Google Meet  6. Participate in the online discussion: Activity No. 6: Mapping Research Agenda  7. Please read this “Exploring the research in information technology implementation for Activity 8  8. Participate in the online discussion: Activity No. 7: Topic Reflection  9. Do this Activity No. 8: Paper Review - Exploring the research in information technology implementation.  Note: The insight that you will post on online discussion forum using Learning Management System (LMS) will receive additional scores in class participation.  You will be given time to complete all assessment tasks and activity provided on the LMS.  1. Watch the video lecture  2. Read the SLM  3. Accomplish the performance tasks:   * Activity No. 6: Mapping Research Agenda [Group Activity] * Activity No. 7: Topic Reflection * Activity No. 8: Topic Reflection and Paper Review - Exploring the research in information technology implementation   (For further instructions, refer to your Google Classroom and see the schedule of activities for this module)  ***Note:*** *The insight that you will post on online discussion forum using Learning Management System (LMS) will receive additional scores in class participation.* |
| **Offline Activities**  **(e-Learning/Self-Paced)** | For offline classes, please refer to the following learning guide questions:  1. Watch the video lecture  2. Read the SLM  3. Accomplish performance tasks using work sheet provided for the following activities:  • Activity No. 6: Mapping Research Agenda [Group Activity]  • Activity No. 7: Topic Reflection  • Activity No. 8: Topic Reflection and Paper Review - Exploring the research in information technology implementation |
| **Module Content** | **Topics Covered:**  Topic 2. Deep Learning for Images  2.1 What is Convolutional Neural Network?  2.2 CNN Architecture  2.3 Training for deep CNN  2.4 Overview of Modern CNN architectures  2.5 Learning new task with CNN  2.6 A glimpse of other Computer Vision tasks  Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. Researchers and enthusiasts alike, work on numerous aspects of the field to make amazing things happen. One of many such areas is the domain of Computer Vision.   1. **What is Convolutional Neural Network**     A CNN sequence in classifying handwritten digit  A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.  The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.  The term ‘Convolution” in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.  For further understanding: [*https://www.youtube.com/watch?v=x\_VrgWTKkiM*](https://www.youtube.com/watch?v=x_VrgWTKkiM)   1. **CNN Architecture**   There are two main parts to a CNN architecture;   * A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction. * A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.     There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.  **Convolution Layers**  Screen-Shot-2018-04-16-at-11.34.51-AM  This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).  The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.  **Pooling Layer**  In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations.  Pooling  In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.  **Fully Connected Layer**  The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.    In this, the input image from the previous layers is flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.  Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.  **Dropout**  Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model’s performance when used on a new data.  To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.  **Activation Functions**  Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.  Activation function  The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. We have different types of activation functions just as in the figure above, but for this post, my focus will be on Rectified Linear Unit (ReLU)  It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax us used.  For further learning: [*https://www.youtube.com/watch?v=YSGihRkpfms*](https://www.youtube.com/watch?v=YSGihRkpfms)   1. **Training for Deep CNN**   Training is the process of taking content that is known to belong to specified classes and creating a classifier on the basis of that known content.  These are the steps used to training the CNN (Convolutional Neural Network).  Steps:  Step 1: Upload Dataset  Step 2: The Input layer  Step 3: Convolutional layer  Step 4: Pooling layer  Step 5: Convolutional layer and Pooling Layer  Step 6: Dense layer  Step 7: Logit Layer  **Upload Dataset**  The MNIST dataset is available with scikit for learning in this URL (Unified Resource Locator). We can download it and store it in our downloads. We can upload it with fetch\_mldata ('MNIST Original').  ***Create a test/train set***  We need to split the dataset into train\_test\_split.  ***Scale the features***  Finally, we scale the function with the help of MinMax Scaler.   1. **import** numpy as np 2. **import** tensorflow as tf 4. from sklearn.datasets **import** fetch\_mldata 5. #Change USERNAME by the username of the machine 6. ##Windows USER 7. mnist = fetch\_mldata('C:\\Users\\USERNAME\\Downloads\\MNIST original') 8. print(mnist.data.shape) 9. print(mnist.target.shape) 10. from sklearn.model\_selection **import** train\_test\_split 11. A\_train, A\_test, B\_train, B\_test = train\_test\_split(mnist.data,mnist.target, test\_size=0.2, random\_state=45) 12. B\_train  = B\_train.astype(**int**) 13. B\_test  = B\_test.astype(**int**) 14. batch\_size =len(X\_train) 15. print(A\_train.shape, B\_train.shape,B\_test.shape ) 16. ## rescale 17. from sklearn.preprocessing **import** MinMaxScaler 18. scaler = MinMaxScaler() 19. # Train the Dataset 20. X\_train\_scaled = scaler.fit\_transform(A\_train.astype(np.float65)) 21. #test the dataset 22. X\_test\_scaled = scaler.fit\_transform(A\_test.astype(np.float65)) 23. feature\_columns = [tf.feature\_column.numeric\_column('x',shape=A\_train\_scaled.shape[1:])] 24. X\_train\_scaled.shape[1:]   **Input Layer**   1. #Input layer 2. def cnn\_model\_fn(mode, features, labels): 3. input\_layer = tf.reshape(tensor= features["x"],shape=[-1, 26, 26, 1])   We need to define a tensor with the shape of the data. For that, we can use the module tf.reshape. In this module, we need to declare the tensor to reshape and to shape the tensor. The first argument is the feature of the data, that is defined in the argument of a function.  A picture has a width, a height, and a channel. The MNIST dataset is a monochromic picture with the 28x28 size. We set the batch size into -1 in the shape argument so that it takes the shape of the features ["x"]. The advantage is to tune the batch size to hyperparameters. If the batch size is 7, the tensor feeds 5,488 values (28 \* 28 \* 7).  **Convolutional Layer**   1. # first CNN Layer 2. conv1 = tf.layers.conv2d( 3. inputs= input\_layer, 4. filters= 18, 5. kernel\_size= [7, 7], 6. padding="same", 7. activation=tf.nn.relu)   The first convolutional layer has 18 filters with the kernel size of 7x7 with equal padding. The same padding has both the output tensor and input tensor have the same width and height. TensorFlow will add zeros in the rows and columns to ensure the same size.  We use the Relu activation function. The output size will be [28, 28, and 14].  **Pooling Layer**  The next step after the convolutional is pooling computation. The pooling computation will reduce the extension of the data. We can use the module max\_pooling2d with a size of 3x3 and stride of 2. We use the previous layer as input. The output size can be [batch\_size, 14, 14, and 15].   1. ##first Pooling Layer 2. pool1 = tf.layers.max\_pooling2d (inputs=conv1, 3. pool\_size=[3, 3], strides=2)   **Pooling Layer and Second Convolutional Layer**  The second CNN has exactly 32 filters, with the output size of [batch\_size, 14, 14, 32]. The size of the pooling layer has the same as ahead, and output shape is [batch\_size, 14, 14, and18].   1. conv2 = tf.layers.conv2d( 2. inputs=pool1, 3. filters=36, 4. kernel\_size=[5, 5], 5. padding="same", 6. activation=tf.nn.relu) 7. pool2 = tf.layers.max\_pooling2d (inputs=conv2, pool\_size=[2, 2],strides=2).   **Fully connected (Dense) Layer**  We have to define the fully-connected layer. The feature map has to be compressed before to be combined with the dense layer. We can use the module reshape with a size of 7\*7\*36.  The dense layer will connect 1764 neurons. We add a Relu activation function and can add a Relu activation function. We add a dropout regularization term with a rate of 0.3, meaning 30 percent of the weights will be 0. The dropout takes place only along the training phase. The cnn\_model\_fn() has an argument mode to declare if the model needs to trained or to be evaluate.   1. pool2\_flat = tf.reshape(pool2, [-1, 7 \* 7 \* 36]) 2. dense = tf.layers.dense(inputs=pool2\_flat, units=7 \* 7 \* 36, activation=tf.nn.relu) 3. dropout = tf.layers.dropout( 4. inputs=dense, rate=0.3, training=mode == tf.estimator.ModeKeys.TRAIN)   **Logits Layer**  Finally, we define the last layer with the prediction of model. The output shape is equal to the batch size 12, equal to the total number of images in the layer.   1. #Logit Layer 2. logits = tf.layers.dense(inputs=dropout, units=12)   We can create a dictionary that contains classes and the possibility of each class. The module returns the highest value with tf.argmax () if the logit layers. The softmax function returns the probability of every class.   1. predictions= { 2. # Generate predictions 3. "classes":tf.argmax(input=logits, axis=1), 4. "probabilities":tf.nn.softmax (logits, name="softmax\_tensor")}   We only want to return the dictionary prediction when the mode is set to prediction. We add these codes to display the predictions.   1. If mode== tf.estimator.ModeKeys.PREDICT: 2. **return** tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)   The next step consists of computing the loss of the model. The loss is easily calculated with the following code:   1. # Calculate Loss (**for** both EVAL and TRAIN modes) 2. loss = tf.losses.sparse\_softmax\_cross\_entropy(labels=labels, logits=logits)   The final step is to optimizing the model, which is to find the best values of weight. For that, we use a gradient descent optimizer with a learning rate of 0.001. The objective is to reduce losses.   1. optimizer= tf.train.GradientDescentOptimizer(learning\_rate=0.0001) 2. train\_op= optimizer.minimize( 3. loss=loss, 4. global\_step=tf.train.get\_global\_step())   We are done with the CNN. However, we want to display the performance metrics during the evaluation mode. The performance metrics for the multiclass model is the accuracy metrics. TensorFlow is equipped with an accuracy model with two arguments, labels, and predicted value.   1. eval\_metric\_ops = { 2. "accuracy": tf.metrics.accuracy(labels=labels, predictions=predictions["classes"])} 3. **return** tf.estimator.EstimatorSpec(mode=mode, loss=loss, eval\_metric\_ops=eval\_metric\_ops)   We can create our first CNN and we are ready to wrap everything in one function to use it and to train and evaluate the model.  A CNN takes many times to training, therefore, we create a logging hook to store the values of the software layers in every 50 iterations.   1. # Set up logging **for** predictions 2. tensors\_to\_log = {"probabilities": "softmax\_tensor"} 3. logging\_hook =tf.train.LoggingTensorHook(tensors=tensors\_to\_log, every\_n\_iter=50)   We are ready to estimator the model. We have a batch size of 100 and shuffle the data into many parts. Note that, we set training steps of 18000, it can take lots of time to train.   1. #Train the model 2. train\_input\_fn = tf.estimator.inputs.numpy\_input\_fn( 3. x={"x": X\_train\_scaled}, 4. y=y\_train, 5. batch\_size=100, 6. num\_epochs=None, 7. shuffle=True) 8. mnist\_classifier.train( 9. input\_fn=train\_input\_fn, 10. steps=18000, 11. hooks=[logging\_hook])   Now, the model is trained, we can evaluate it and print the results easily.   1. # Evaluate the model and print the results 2. eval\_input\_fn = tf.estimator.inputs.numpy\_input\_fn( 3. x= {"x": X\_test\_scaled}, 4. y=y\_test, 5. num\_epochs=1, 6. shuffle=False) 7. eval\_results = mnist\_classifier.evaluate(input\_fn=eval\_input\_fn) 8. print(eval\_results)   **CNN Creation:** [*https://www.youtube.com/watch?v=WvoLTXIjBYU*](https://www.youtube.com/watch?v=WvoLTXIjBYU)  **Overview of Modern CNN Architecture**  CNN Architecture over a timeline Over the years, CNNs have undergone a considerable amount of rework and advancement. This has left us with a plethora of CNN models. Let’s discuss the more important CNNs out of all these variants.LeNet-5 **Architecture**: LeNet-5 has 2 convolutional and 3 fully connected layers. It has trainable weights and a sub-sampling layer (now known as the pooling layer). LeNet5 has about 60,000 parameters.  **Year of Release**: 1998  **About:**Developed by Yann LeCunn as he applied a backdrop style to Fukushima’s convolutional neural network architecture.  **USP:** LeNet5 can be considered the **standard template** for all modern CNNs as all CNNs follow the pattern of stacking convolutional and pooling layers, and terminating the model with one or more fully-connected layers. AlexNet **Architecture**: AlexNet has 8 layers, 3 fully-connected and 5 convolutional. AlexNet had 60 million parameters.  **Year of Release**: 2012  **About:** On the date of its publication, the authors of AlexNet believed that it was the largest neural network on the subsets of ImageNet.  **USP:** AlexNet developers successfully used overlapping pooling and Rectified Linear Units (ReLUs, as activation functions). VGG-16 **Architecture**: VGG-16 has 13 convolutional and 3 fully-connected layers. It used ReLUs as activation functions, just like in AlexNet. VGG-16 had 138 million parameters. A deeper version, VGG-19, was also constructed along with VGG-16.  **Year of Release**: 2014  **About**: Believing that the best way to improve the efficiency of a CNN was to stack more layers onto it, developers at Visual Geometry Group (VGG) developed VGG-16 and VGG-19.  **USP**: First among the deeper CNNs. Inception-v1 **Architecture**: Inception-v1 heavily used the Network in Network approach and had  22 layers along with 5 million parameters.  **Year of Release**: 2014  **About**: This network was a result of a study on approximating sparse architectures. The strongest feature of this network was the improved usage of computer resources inside the neural network.  **USP**: Instead of stacking convolutional layers atop each other, this network stacked dense modules which had convolutional layers within them. Inception-v3 **Architecture**: A successor to Inception-v1, Inception v-3 had 24 million parameters and ran 48 layers deep.  **Year of Release**: 2015  **About:** Inception v3 could classify images into a total of 1000 categories, including keyboard, pencil, mouse, and many other animals. This model was trained on more than one million images from the ImageNet database.  **USP:**Inception v3 was among the first algorithms to use batch normalization. It also used the factorization method to have more efficient computations. ResNet-50 **Architecture:**consisting of 50 layers of ResNet blocks (each block having 2 or 3 convolutional layers), ResNet 50 had 26 million parameters.  **Year of Release**: 2015  **About:**The basic building blocks for ResNet-50 are convolutional and identity blocks. To address the degradation in accuracy, Microsoft researchers added skip connection ability.  **USP**: ResNet-50 popularized skip connection and provided a way for developers to build even deeper CNNs without compromising accuracy. Also, ResNet-50 was among the first CNNs to have the batch normalization feature. Xception **Architecture:** Xception was 71 layers deep and had 23 million parameters. It was based on Inception-v3.  **Year of Release**: 2016  **About:** Xception was heavily inspired by Inception-v3, albeit it replaced convolutional blocks with depth-wise separable convolutions.  **USP**: Xception practically is a CNN based solely on depth-wise separable convolutional layers Inception-v4 **Architecture**: With 43 million parameters and an upgraded Stem module, Inception-v4 is touted to have a dramatically improved training speed due to residual connections.  **Year of Release**: 2016  **About:** Developed by Google researcher, Inception v4 had undergone uniform choices for each grid size.  **USP**: deeper network, Stem improvements, and the same number of filters in every convolution block. Inception-ResNets **Architecture:**The Inception-ResNet had 25 million parameters and 32 towers.  **Year of Release**: 2017  **About**: It was a combination of Inception v4 and ResNet-50.  **USP**: Scaled up cardinality within a module. ResNeXt-50 Architecture: At 50 layers deep and sporting 25.5 million parameters, ResNeXt-50 was trained on more than a million images from the ImageNet dataset.  Year of Release: 2017  About: An improvement over ResNet, ResNeXt-50 displayed a 3.03% error rate with a considerable relative improvement of 15%.  USP: Scaled up the cardinality dimension. A Glimpse of Other Computer Vision Task The following computer vision problems where deep learning has been used:   1. Image Classification with Localization 2. Object Detection 3. Object Segmentation 4. Image Style Transfer 5. Image Colorization 6. Image Reconstruction 7. Image Super-Resolution 8. Image Synthesis 9. Other Problems   **Image Classification with Localization**  Image classification with localization involves assigning a class label to an image and showing the location of the object in the image by a bounding box (drawing a box around the object).  This is a more challenging version of image classification.  Some examples of image classification with localization include:   * Labeling an x-ray as cancer or not and drawing a box around the cancerous region. * Classifying photographs of animals and drawing a box around the animal in each scene.   **Object Detection**  Object detection is the task of image classification with localization, although an image may contain multiple objects that require localization and classification.  This is a more challenging task than simple image classification or image classification with localization, as often there are multiple objects in the image of different types.  Often, techniques developed for image classification with localization are used and demonstrated for object detection.  Some examples of object detection include:   * Drawing a bounding box and labeling each object in a street scene. * Drawing a bounding box and labeling each object in an indoor photograph. * Drawing a bounding box and labeling each object in a landscape.   **Object Segmentation**  Object segmentation, or semantic segmentation, is the task of object detection where a line is drawn around each object detected in the image. Image segmentation is a more general problem of spitting an image into segments.  Object detection is also sometimes referred to as object segmentation.  Unlike object detection that involves using a bounding box to identify objects, object segmentation identifies the specific pixels in the image that belong to the object. It is like a fine-grained localization.  **Image Style Transfer**  Style transfer or neural style transfer is the task of learning style from one or more images and applying that style to a new image.  This task can be thought of as a type of photo filter or transform that may not have an objective evaluation.  Examples include applying the style of specific famous artworks (e.g. by Pablo Picasso or Vincent van Gogh) to new photographs.  Datasets often involve using famous artworks that are in the public domain and photographs from standard computer vision datasets.  **Image Colorization**  Image colorization or neural colorization involves converting a grayscale image to a full color image.  This task can be thought of as a type of photo filter or transform that may not have an objective evaluation.  Examples include colorizing old black and white photographs and movies.  Datasets often involve using existing photo datasets and creating grayscale versions of photos that models must learn to colorize.  **Image Reconstruction**  Image reconstruction and image inpainting is the task of filling in missing or corrupt parts of an image.  This task can be thought of as a type of photo filter or transform that may not have an objective evaluation.  Examples include reconstructing old, damaged black and white photographs and movies (e.g. photo restoration).  Datasets often involve using existing photo datasets and creating corrupted versions of photos that models must learn to repair.  **Image Super-Resolution**  Image super-resolution is the task of generating a new version of an image with a higher resolution and detail than the original image.  Often models developed for image super-resolution can be used for image restoration and inpainting as they solve related problems.  Datasets often involve using existing photo datasets and creating down-scaled versions of photos for which models must learn to create super-resolution versions.  **Image Synthesis**  Image synthesis is the task of generating targeted modifications of existing images or entirely new images.  This is a very broad area that is rapidly advancing.  It may include small modifications of image and video (e.g. image-to-image translations), such as:   * Changing the style of an object in a scene. * Adding an object to a scene. * Adding a face to a scene.   **Other Problems**  There are other important and interesting problems that I did not cover because they are not purely computer vision tasks.  Notable examples image to text and text to image:   * Image Captioning: Generating a textual description of an image. * Show and Tell: A Neural Image Caption Generator, 2014. * Image Describing: Generating a textual description of each object in an image. * Deep Visual-Semantic Alignments for Generating Image Descriptions, 2015. * Text to Image: Synthesizing an image based on a textual description. * AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks, 2017.   Presumably, one learns to map between other modalities and images, such as audio. |

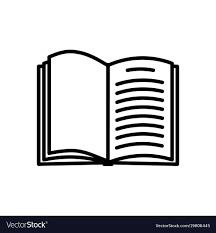
****

**Performance Tasks**

|  |
| --- |
| **For the activities:**  The line of codes given in the *Training for Deep CNN* topic was supposed to run using google colab. Your task is to make sure that the given code will run using google colab with your chosen dataset and have at least 90% accuracy rate in prediction. You are allowed to used other sources such as projects in github. Note that the higher the accuracy the higher rate you will have. Further, to complete this task you are required to submit the screenshot of every layer upon running and give at least 5 sentences explanation on how did you get your accuracy. Good luck ! |

**Understanding Directed Assess**

|  |
| --- |
|  |

 **Learning Resources**

|  |
| --- |
| * <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html> * <https://sites.google.com/site/machinelearningnotebook2/classification/multi-class-classification/backpropagation> * <https://www.coursera.org/learn/intro-to-deep-learning?specialization=aml> |