

MIT ART, DESIGN AND TECHNOLOGY UNIVERSITY

MIT SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

COURSE NAME:

DEEP LEARNING & NEURAL NETWORKS LAB

COURSE CODE:

21BTCS042

CLASS: LY

BRANCH: CSE

SPECIALIZATION: AIA

PREPARED BY: DR. NAGESH JADHAV

REQUIRED H/W & S/W: 64 bit processor based machine,
Anaconda, Python, Visual Studio Code, Spyder IDE

MIT Art, Design and Technology University

VISION

MIT Art, Design and Technology University aspires to be the University of Eminence by amalgamating Art, Design, Science and Technology. The University aims to have a transformative impact on society through holistic education, multidisciplinary research ethos, innovation and entrepreneurial culture.

MISSION

- The Mission of MIT Art, Design and Technology University is to provide impetus to faculty, learners, and staff by developing their innate intellectual capabilities, creative abilities and entrepreneurial mind-set for the socio-economic development of the nation.
- We empower learners to become adaptive and agile global professionals through unique specialized programs building academia-industrial partnership.
- We nurture learners to be intellectually curious, technologically equipped, mentally sound, physically fit, spiritually elevated, socio-culturally sensitive,
- environmentally conscious through continuous holistic education for the ever- evolving world.
- We provide technology-enabled learner-driven curriculum, value added courses, simulated learning environments, state-of-the-art infrastructure and opportunities for community engagement.

School of Engineering

VISION

The School of Engineering aims to achieve transformative impact through academic excellence and innovation for socio-sustainable development.

MISSION

- To disseminate knowledge in computing and multidisciplinary domains with specialized and need-based curriculum, innovating teaching pedagogy and state-of- art infrastructure.
- To develop a culture for high-impact research, innovation and entrepreneurship by creating knowledge through design thinking and project- based learning.
- To imbibe value-based and holistic education for next-generation professionals toward societal development and nation-building.
- To promote conducive work-culture for capacity building of the employees.

Department of Computer Science & Engineering

VISION

The Department of Computer Science Engineering aims to impart progressive education with a state-of-the-art curriculum and generate socially sensitized engineers, innovators, and entrepreneurs for sustainable development.

MISSION

M-1: To pioneer the education in Computer Science and Engineering with a dynamic and industry-ready curriculum.

M-2: To evolve as a Centre of Excellence that contributes significantly to multi-disciplinary research, innovation and entrepreneurship in collaboration with industry and academia.

M-3: To encourage students and faculties by engaging in multi-disciplinary problem-solving and creating sustainable solutions for social well-being.

M-4: To nurture students with ethics and values through holistic education for developing responsible leadership.

Program Educational Objective (PEO's)

PEO1-To apply the acquired knowledge of Computer Science & Engineering to build sustainable solutions to real-world problems in society.

PEO2-Attain the ability to adapt quickly to new environments and technologies, assimilate new information, and work in multi-disciplinary areas with a strong focus on higher studies, innovation and entrepreneurship.

PEO3-To practice collaborative learning and team spirit as professionals with project-based learning and technical activities.

PEO4-To be able to effectively present, communicate and exhibit professionalism with ethical values and empathy for needs of society.

Program Outcome (PO's)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Lifelong learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

Deep Learning & Neural Network Lab			
Course Code:	21BTCS042	Course Credits:	02
Teaching Hours / Week (L:T:P):	0:0:4	CA Marks:	20
Total Number of Teaching Hours:	60	END-SEM Marks:	30
Course Pre-requisites: <ul style="list-style-type: none"> Python programming, Machine Learning 			
Course Description: <ul style="list-style-type: none"> This course provides an introduction to deep learning. It starts with configuration settings to execute machine learning experiments. It covers frameworks like tensorflow, pytorch etc. Assigned are designed to cover entire cycle of deep learning process. The advanced topics like introduction to autoencoders and image processing is also covered in this course. 			
Course Learning Objectives: This course will enable the students to: <ol style="list-style-type: none"> To introduce students to the basic concepts and techniques of deep Learning. To become familiar with CNN and sequential modeling. To learn and understand autoencoders. 			
Course Outcomes: At the end of the course the student shall be able to <ol style="list-style-type: none"> To identify and apply deep learning techniques to solve real world problems To apply various optimizers and fine tune hyperparameters. To apply and implement CNN for feature extraction and classification on image data. To apply and implement RNN for sequential and time series dataset To apply advanced deep learning techniques for dimensionality reduction & feature extraction. 			
Text Books: <ul style="list-style-type: none"> Ian Goodfellow and Yoshua Bengio, Deep Learning (Adaptive Computation and machine Learning Series), Massachusetts London, England, ISBN No. 9780262035613. Nikhil Buduma, Fundamentals of Deep Learning, O'Reilly, First Edition, ISBN No. 978-14-919- 2561-4. 			
Reference Books: <ul style="list-style-type: none"> Christopher M. Bishop, Pattern Recognition and Machine Learning, McGraw-Hill, ISBN No. 0- 07- 115467-1. 			
Online references: <ol style="list-style-type: none"> https://onlinecourses.nptel.ac.in/ 			

Sr. No.	Name of Experiment/Assignment	CO
1.	Introduction to Keras and Tensorflow (Optional – Pytorch). Configure and use google colab and kaggle GPU.	CO-I
2.	Develop multi class classifier using deep multilayer perceptron (Keras/tensorflow/pytorch) for MNIST hand recognition dataset and CIFAR10. Fine the parameters for better accuracy. Analyse the model accuracy and generate classification report. Plot accuracy and loss graph. <ul style="list-style-type: none"> Develop application with GUI to upload input to the system. Test the model. 	CO-I
3.	Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report. <ul style="list-style-type: none"> Develop an GUI and test the user given inputs. Analyze the result with and without regularization/dropout 	CO-III
4.	Develop face recognition system using CNN. Create a dataset of minimum 50 students from your class. Check and validate the accuracy of the model. <ul style="list-style-type: none"> Apply dimensionality reduction on input image and plot the change in accuracy of system. 	CO-III
5.	Write a program to demonstrate the change in accuracy/loss/convergence time with change in optimizers like stochastic gradient descent, adam, adagrad, RMSprop and Nadam for any suitable application	CO-II
6.	Apply transfer learning with pre-trained VGG16/ResNet50/MobileNet model on given dataset and analyze the results.	CO-III
7.	Develop RNN/LSTM/GRU model for suitable application.	CO-IV
8.	Develop an autoencoder to encode and decode the image. Analyse the results.	CO-V
9.	Case study- GAN	CO-V

ASSESSMENT AND EVALUATION PATTERN		
WEIGHTAGE	Continuous Assessment (CA)	End Semester Assessment (ESA)
CA - Assignments (PBL / Case Study / Presentation / Seminar / Group Discussions / Quiz / Test		
ESA - Bloom's Taxonomy Levels: Remembering, Understanding, Applying, Analyzing, Evaluating, and Creating		
Experiential Learning - Internship (Summer / Winter) / Industry Visit / Site Visit / Field Trips		
Project Based Learning – Mini (Minor) Project / Major Project		

Course Articulation Matrix (CO-PO Mapping) [Deep Learning & Neural Network Lab–21BTCS042]

[illegible]

Sr. No.	Name of Experiment	CO
1.	Introduction to Keras and Tensorflow (Optional – Pytorch). Configure and use google colab and kaggle GPU.	CO-I
2.	<p>Develop multi class classifier using deep multilayer perceptron (Keras/tensorflow/pytorch) for MNIST hand recognition dataset and CIFAR10. Fine the parameters for better accuracy. Analyse the model accuracy and generate classification report. Plot accuracy and loss graph.</p> <ul style="list-style-type: none"> Develop application with GUI to upload input to the system. Test the model. 	CO-I
3.	<p>Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report.</p> <ul style="list-style-type: none"> Develop an GUI and test the user given inputs. Analyze the result with and without regularization/dropout 	CO-III
4.	<p>Develop face recognition system using CNN. Create a dataset of minimum 50 students from your class. Check and validate the accuracy of the model.</p> <ul style="list-style-type: none"> Apply dimensionality reduction on input image and plot the change in accuracy of system. 	CO-III
5.	Write a program to demonstrate the change in accuracy/loss/convergence time with change in optimizers like stochastic gradient descent, adam, adagrad, RMSprop and Nadam for any suitable application	CO-II
6.	Apply transfer learning with pre-trained VGG16/ResNet50/MobileNet model on given dataset and analyze the results.	CO-III
7.	Develop RNN/LSTM/GRU model for suitable application.	CO-IV
8.	Develop an autoencoder to encode and decode the image. Analyse the results.	CO-V
9.	Case study- GAN	CO-V

Assignment No. 1

Aim: Introduction to Keras and Tensorflow (Optional – Pytorch). Configure and use google colab and kaggle GPU

Objectives:

1. To configure anaconda and google colab, kaggle environment
2. To Explore TF/Keras/Pytorch libraries
3. To learn to use GPU/TPU
4. To learn and understand Git

Theory:

Keras Configuration

1. Setup Environment
2. pip install keras
3. to check whether it has installed properly: python>>import keras

Python>> print keras.__version__

Tensorflow Configuration:

1. pip install tensorflow==2.2.0
2. To verify installation: python>> import tensorflow as tf

If no error then the installation has been completed

Colaboratory Configuration

1. Setup the environment
2. Connect to the Drive
3. Upload the files using: from google.colab import files
files.upload()

Kaggle Configuration:

1. Create a Kaggle Account
2. Create an Authorization token
3. Upload on Colab using the upload code
4. Make a folder and make the Json file executable
5. Get the dataset

6. Copy the API command and run on colab

GitHub Configuration

1. pip install git
2. Set up user profile by: global user.name "name"
Global user.email "mail"

Dataset Attributes:

1. Pregnancies
2. Glucose
3. BloodPressure
4. SkinThickness
5. Insulin
6. BMI
7. DiabetesPedigreeFunction
8. Age
9. Outcome

Code:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df=pd.read_csv("diabetes.csv")
x=df.iloc[:,0:8]
y=df["Outcome"]
obj=StandardScaler()
x=obj.fit_transform(x)
Xtrain,Xtest,Ytrain,Ytest=train_test_split(x,y,test_size=0.1)

model=models.Sequential()
model.add(layers.Dense(100,activation="relu"))
```

```
#model.add(layers.Dense(75,activation="relu"))
model.add(layers.Dense(50,activation="relu"))
#model.add(layers.Dense(25,activation="relu"))
model.add(layers.Dense(12,activation="relu"))
model.add(layers.Dense(8,activation="relu"))
model.add(layers.Dense(1,activation="sigmoid"))

model.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"],)
history=model.fit(Xtrain,Ytrain,epochs=50, validation_data=(Xtest,Ytest))
result=model.evaluate(Xtest,Ytest)
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], label='loss')
#plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('loss')
plt.ylim([0, 0.8])
plt.legend(loc='lower right')
```

```
test_loss, test_acc = model.evaluate(Xtest, Ytest, verbose=2)
plt.ylim([0.6,1])
plt.plot(history.history['accuracy'], label = 'accuracy')
```

Results:

Training:

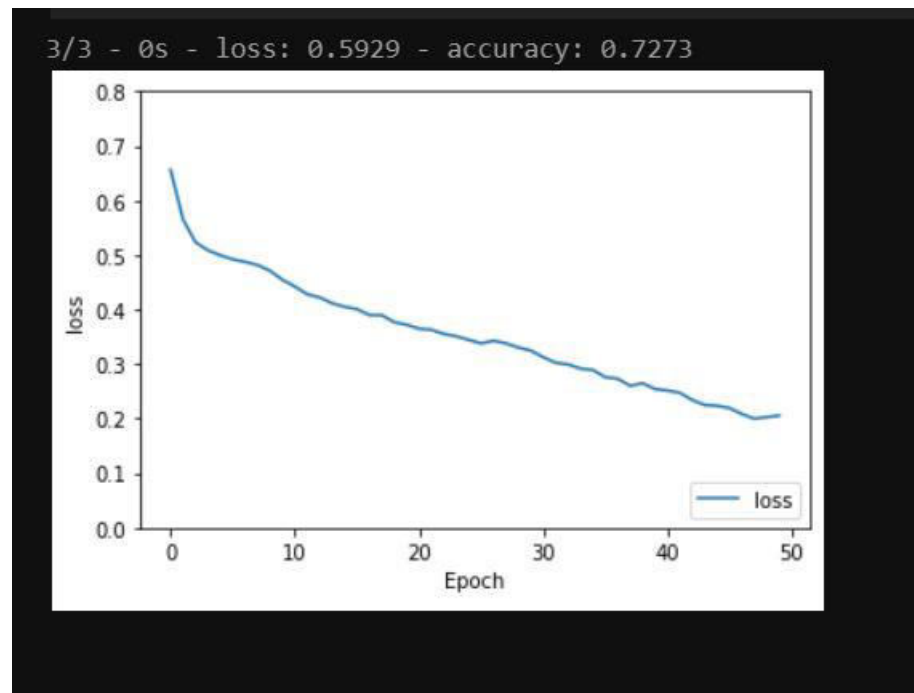
```
Epoch 1/50
22/22 [=====] - 1s 23ms/step - loss: 0.6561 - accuracy: 0.6483 - val_loss: 0.5546 - val_accuracy: 0.6753
Epoch 2/50
22/22 [=====] - 0s 6ms/step - loss: 0.5653 - accuracy: 0.6483 - val_loss: 0.4855 - val_accuracy: 0.6753
Epoch 3/50
22/22 [=====] - 0s 6ms/step - loss: 0.5234 - accuracy: 0.6483 - val_loss: 0.4770 - val_accuracy: 0.6753
Epoch 4/50
22/22 [=====] - 0s 7ms/step - loss: 0.5087 - accuracy: 0.6527 - val_loss: 0.4726 - val_accuracy: 0.7403
Epoch 5/50
22/22 [=====] - 0s 7ms/step - loss: 0.4994 - accuracy: 0.7496 - val_loss: 0.4736 - val_accuracy: 0.8052
Epoch 6/50
22/22 [=====] - 0s 9ms/step - loss: 0.4920 - accuracy: 0.7974 - val_loss: 0.4664 - val_accuracy: 0.8182
Epoch 7/50
22/22 [=====] - 0s 7ms/step - loss: 0.4873 - accuracy: 0.7902 - val_loss: 0.4674 - val_accuracy: 0.8312
Epoch 8/50
22/22 [=====] - 0s 7ms/step - loss: 0.4815 - accuracy: 0.7916 - val_loss: 0.4685 - val_accuracy: 0.8052
Epoch 9/50
22/22 [=====] - 0s 7ms/step - loss: 0.4711 - accuracy: 0.7916 - val_loss: 0.4599 - val_accuracy: 0.8312
Epoch 10/50
22/22 [=====] - 0s 7ms/step - loss: 0.4546 - accuracy: 0.7931 - val_loss: 0.4447 - val_accuracy: 0.8052
Epoch 11/50
22/22 [=====] - 0s 7ms/step - loss: 0.4422 - accuracy: 0.7873 - val_loss: 0.4392 - val_accuracy: 0.7922
Epoch 12/50
22/22 [=====] - 0s 7ms/step - loss: 0.4281 - accuracy: 0.8032 - val_loss: 0.4383 - val_accuracy: 0.8182
Epoch 13/50
```

```
22/22 [=====] - 0s 7ms/step - loss: 0.2601 - accuracy: 0.9030 - val_loss: 0.4941 - val_accuracy: 0.7792
Epoch 39/50
22/22 [=====] - 0s 7ms/step - loss: 0.2649 - accuracy: 0.9045 - val_loss: 0.4924 - val_accuracy: 0.7532
Epoch 40/50
22/22 [=====] - 0s 7ms/step - loss: 0.2543 - accuracy: 0.9074 - val_loss: 0.5258 - val_accuracy: 0.7532
Epoch 41/50
22/22 [=====] - 0s 8ms/step - loss: 0.2515 - accuracy: 0.8958 - val_loss: 0.5441 - val_accuracy: 0.7143
Epoch 42/50
22/22 [=====] - 0s 7ms/step - loss: 0.2473 - accuracy: 0.9103 - val_loss: 0.5321 - val_accuracy: 0.7403
Epoch 43/50
22/22 [=====] - 0s 10ms/step - loss: 0.2341 - accuracy: 0.9204 - val_loss: 0.5547 - val_accuracy: 0.7273
Epoch 44/50
22/22 [=====] - 0s 9ms/step - loss: 0.2249 - accuracy: 0.9233 - val_loss: 0.5368 - val_accuracy: 0.7792
Epoch 45/50
22/22 [=====] - 0s 10ms/step - loss: 0.2236 - accuracy: 0.9190 - val_loss: 0.5579 - val_accuracy: 0.7403
Epoch 46/50
22/22 [=====] - 0s 8ms/step - loss: 0.2193 - accuracy: 0.9204 - val_loss: 0.5855 - val_accuracy: 0.7143
Epoch 47/50
22/22 [=====] - 0s 9ms/step - loss: 0.2084 - accuracy: 0.9305 - val_loss: 0.5877 - val_accuracy: 0.7273
Epoch 48/50
22/22 [=====] - 0s 8ms/step - loss: 0.1997 - accuracy: 0.9334 - val_loss: 0.5976 - val_accuracy: 0.7013
Epoch 49/50
22/22 [=====] - 0s 6ms/step - loss: 0.2029 - accuracy: 0.9363 - val_loss: 0.5827 - val_accuracy: 0.7143
Epoch 50/50
22/22 [=====] - 0s 7ms/step - loss: 0.2054 - accuracy: 0.9334 - val_loss: 0.5929 - val_accuracy: 0.7273
```

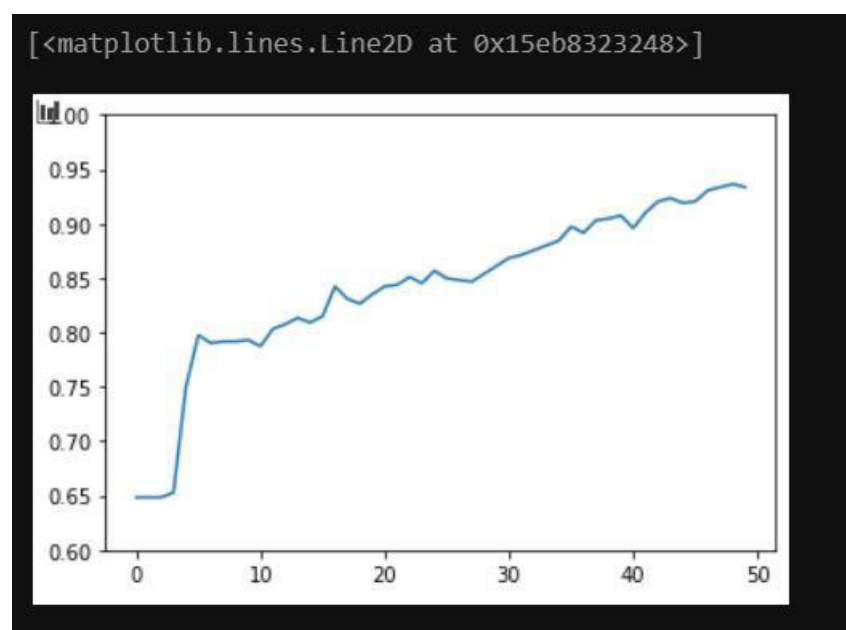
Evaluation

```
MI
result=model.evaluate(Xtest,Ytest)
3/3 [=====] - 0s 3ms/step - loss: 0.5929 - accuracy: 0.7273
```

Loss function



Accuracy



Conclusion:

Thus we have understood the configuration steps of Google Colab, tensorflow, etc and learned to use tensorflow and create a model to predict the chance of diabetes or not.

Assignment No. 2

Aim: Develop multi class classifier using deep multilayer perceptron (Keras/tensorflow/pytorch) for MNIST hand recognition dataset and CIFAR10. Fine the parameters for better accuracy.

☑ Develop application with GUI to upload input to the system.

☑ Test the model

Objectives:

1. Learn Deep Neural Network modeling
2. Learn to develop and deploy models

Theory:

Standardisation

This is one of the most use type of scalar in data preprocessing . This is known as z-score . This re distribute the data in such a way that mean (μ) = 0 and standard deviation (σ) =1 . Here is the below formula for calculation

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$$

Normalization:

Normalization scales the feature between 0.0 & 1.0, retaining their proportional range to each other.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

The range of normal distribution is [-1,1] with mean =0.

Data Splitting

Train Test Split is one of the important steps in Machine Learning. It is very important because your model needs to be evaluated before it has been deployed. And that evaluation needs to be done on unseen data because when it is deployed, all incoming data is unseen.

The main idea behind the train test split is to convert original data set into 2 parts

- train
- test

where train consists of training data and training labels and test consists of testing data and testing labels.

The easiest way to do it is by using *scikit-learn*, which has a built-in function *train_test_split*

Data Cleaning

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted.

This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide inaccurate results. There are several methods for cleaning data depending on how it is stored along with the answers being sought.

Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information.

For one, data cleaning includes more actions than removing data, such as fixing spelling and syntax errors, standardizing data sets, and correcting mistakes such as empty fields, missing codes, and identifying duplicate data points. Data cleaning is considered a foundational element of the data science basics, as it plays an important role in the analytical process and uncovering reliable answers.

Code:

For MNIST

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import models, datasets, layers
import matplotlib.pyplot as plt
import matplotlib.image as mp
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
train_images = train_images / 255
test_images = test_images / 255
```

```

model=models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28, 1)))
model.add(layers.Dense(32,activation="relu"))
model.add(layers.Dense(16,activation="relu"))
model.add(layers.Dense(10,activation="softmax"))

model.compile(optimizer='adam',loss="sparse_categorical_crossentropy",metrics=
['accuracy'])

model.fit(train_images, train_labels, epochs=10,validation_data=(test_images,t
est_labels))

```

For CIFAR

```

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import models,layers,datasets

(train_images, train_labels),(test_images,test_labels)=datasets.cifar10.load_d
ata()
train_images=train_images/255
test_images=test_images/255

model=models.Sequential()
model.add(layers.Flatten(input_shape=(32, 32, 3)))
model.add(layers.Dense(512,activation="relu"))
#model.add(layers.Dense(256,activation="relu"))
model.add(layers.Dense(128,activation="relu"))
#model.add(layers.Dense(64,activation="relu"))
model.add(layers.Dense(32,activation="relu"))
#model.add(layers.Dense(16,activation="relu"))
model.add(layers.Dense(10,activation="softmax"))

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

model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,tes
t_labels))

```

MNIST GUI

```

from PIL import Image,ImageOps
import os
import streamlit as st

```

```

import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras.preprocessing import image
import numpy as np
st.set_option('deprecation.showfileUploaderEncoding', False)

def load_models(img):
    model = models.load_model('/tmp/model')
    image=img.resize((28,28))
    image_array=np.array(image)
    image_array=tf.image.rgb_to_grayscale(image_array)
    image_array=(tf.reshape(image_array,(image_array.shape[0],image_array.shap
e[0],1)))/255
    image_array=np.array([image_array])

    prediction = model.predict_classes(image_array)
    return prediction

def upload_images():
    uploaded_file = st.file_uploader("Choose an Image ...", type="jpg")
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded The image.', use_column_width=True)
        st.write("")
        st.write("Classifying...")
        label = load_models(image)

        if label==0:
            st.write("0")
        if label==1:
            st.write("1")
        if label==2:
            st.write("2")
        if label==3:
            st.write("3")
        if label==4:
            st.write("4")
        if label==5:
            st.write("5")
        if label==6:
            st.write("6")
        if label==7:
            st.write("7")
        if label==8:
            st.write("8")
        if label==9:
            st.write("9")

```



```

if __name__ == "__main__":
    st.header("MNIST DATA classification")
    st.write("Upload an image")

    upload_images()

```

CIFAR GUI

```

from PIL import Image, ImageOps
import os
import streamlit as st
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras.preprocessing import image
import numpy as np
st.set_option('deprecation.showfileUploaderEncoding', False)
def load_models(img):
    model = models.load_model('/tmp/model1')
    image=img.resize((32,32))
    image_array=np.array(image)
    #image_array=tf.image.rgb_to_grayscale(image_array)
    image_array=(tf.reshape(image_array,(image_array.shape[0],image_array.shap
e[0],3)))/255
    image_array=np.array([image_array])

    prediction = model.predict_classes(image_array)
    return prediction

def upload_images():
    uploaded_file = st.file_uploader("Choose an Image ...", type="jpg")
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded The image.', use_column_width=True)
        st.write("")
        st.write("Classifying...")
        label = load_models(image)

        if label==0:
            st.write("It is an aeroplane")
        if label==1:
            st.write("It is an automobile")
        if label==2:
            st.write("It is a bird")
        if label==3:

```

```

        st.write("It is an cat")
    if label==4:
        st.write("It is a deer")
    if label==5:
        st.write("It is a dog")
    if label==6:
        st.write("It is a frog")
    if label==7:
        st.write("It is a horse")
    if label==8:
        st.write("It is a ship")
    if label==9:
        st.write("It is a truck")

if __name__ == "__main__":
    st.header("CIFAR DATA classification(DENSE LAYERS)")
    st.write("Upload an image")

    upload_images()

```

Results:

MNIST Training:

```

> MI
model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
Epoch 1/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.3898 - accuracy: 0.8856 - val_loss: 0.2218 - val_accuracy: 0.9343
Epoch 2/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.1935 - accuracy: 0.9452 - val_loss: 0.1641 - val_accuracy: 0.9525
Epoch 3/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.1504 - accuracy: 0.9564 - val_loss: 0.1411 - val_accuracy: 0.9583
Epoch 4/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.1271 - accuracy: 0.9622 - val_loss: 0.1380 - val_accuracy: 0.9589
Epoch 5/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.1112 - accuracy: 0.9668 - val_loss: 0.1158 - val_accuracy: 0.9663
Epoch 6/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.0979 - accuracy: 0.9704 - val_loss: 0.1184 - val_accuracy: 0.9651
Epoch 7/10
1875/1875 [=====] - 7s 4ms/step - loss: 0.0883 - accuracy: 0.9729 - val_loss: 0.1155 - val_accuracy: 0.9668
Epoch 8/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.0796 - accuracy: 0.9752 - val_loss: 0.1185 - val_accuracy: 0.9668
Epoch 9/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0735 - accuracy: 0.9773 - val_loss: 0.1313 - val_accuracy: 0.9627
Epoch 10/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.0686 - accuracy: 0.9788 - val_loss: 0.1322 - val_accuracy: 0.9642
<tensorflow.python.keras.callbacks.History at 0x18009518ac8>

```

CIFAR10 Training

```
ML
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels))

Epoch 1/10
1563/1563 [=====] - 24s 15ms/step - loss: 1.9808 - accuracy: 0.2660 - val_loss: 1.8042 - val_accuracy: 0.3416
Epoch 2/10
1563/1563 [=====] - 24s 15ms/step - loss: 1.7372 - accuracy: 0.3699 - val_loss: 1.6930 - val_accuracy: 0.3866
Epoch 3/10
1563/1563 [=====] - 24s 15ms/step - loss: 1.6558 - accuracy: 0.4044 - val_loss: 1.6261 - val_accuracy: 0.4132
Epoch 4/10
1563/1563 [=====] - 24s 15ms/step - loss: 1.5997 - accuracy: 0.4209 - val_loss: 1.5775 - val_accuracy: 0.4325
Epoch 5/10
1563/1563 [=====] - 25s 16ms/step - loss: 1.5563 - accuracy: 0.4387 - val_loss: 1.5287 - val_accuracy: 0.4481
Epoch 6/10
1563/1563 [=====] - 25s 16ms/step - loss: 1.5220 - accuracy: 0.4516 - val_loss: 1.4918 - val_accuracy: 0.4695
Epoch 7/10
1563/1563 [=====] - 25s 16ms/step - loss: 1.4832 - accuracy: 0.4669 - val_loss: 1.4855 - val_accuracy: 0.4665
Epoch 8/10
1563/1563 [=====] - 25s 16ms/step - loss: 1.4591 - accuracy: 0.4777 - val_loss: 1.4884 - val_accuracy: 0.4603
Epoch 9/10
1563/1563 [=====] - 25s 16ms/step - loss: 1.4357 - accuracy: 0.4842 - val_loss: 1.4675 - val_accuracy: 0.4717
Epoch 10/10
1563/1563 [=====] - 25s 16ms/step - loss: 1.4125 - accuracy: 0.4918 - val_loss: 1.4556 - val_accuracy: 0.4761

<tensorflow.python.keras.callbacks.History at 0x25e81272588>
```

MNIST GUI

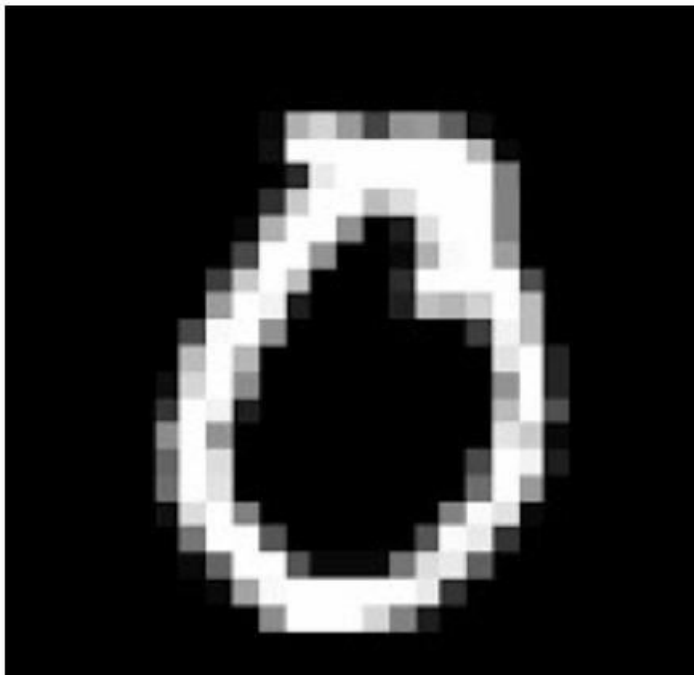
MNIST DATA classification

Upload an image

Choose an Image ...

mnist_manual_input.jpg

browse files



Uploaded The image.

Classifying...

0

CIFAR10 GUI

CIFAR DATA classification(DENSE LAYERS)

Upload an image

Choose an Image ...

white-moving-truck-parked-alley-61314277.jpg

[browse files](#)



Uploaded The image.

Classifying...

It is a truck

Conclusion:

Thus, we have understood the syntax and basic model creation in TensorFlow for 2 different task.

We have also learned how to create a GUI using services to do so.

Assignment No. 2

Aim: Develop classification model for cat-dogs dataset using CNN model. Analyze the model accuracy and generate classification report.

□ Develop an application and test the user given inputs.

Analyze the result with and without regularization/dropout

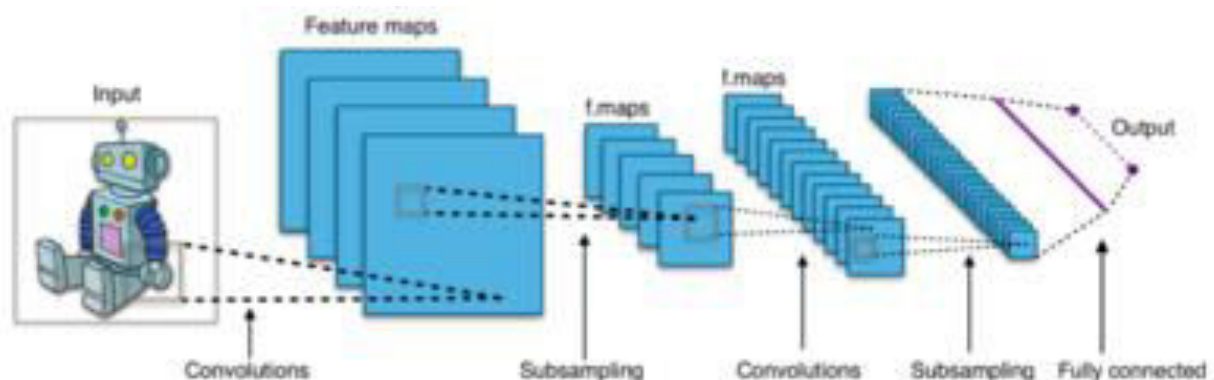
Objectives:

1. To learn about classification
2. To learn CNN
3. To demonstrate and analyse the results

Theory:

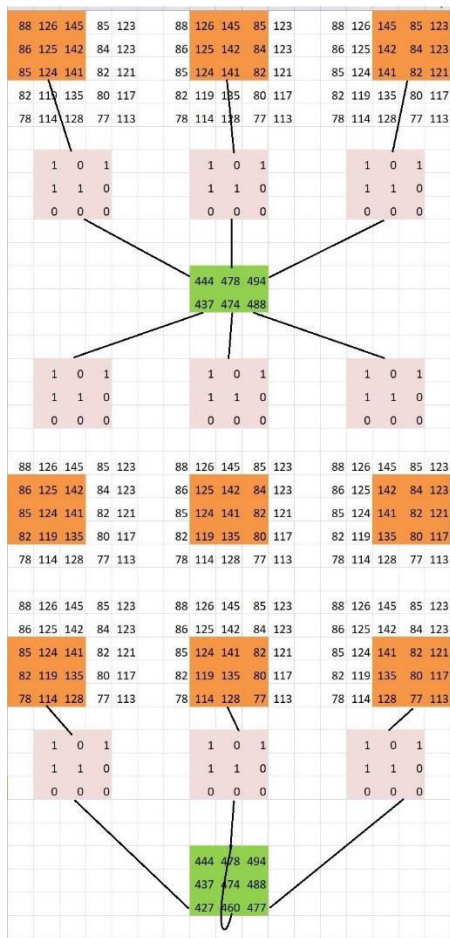
A **convolutional neural network (CNN, or ConvNet)** is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as **shift invariant** or **space invariant artificial neural networks (SIANN)**, based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.



Mathematics

Convolution Process



Local Receptive Field

Filter

Output image

Output image value = LRF * Filter
(dot product of LRF and Filter)

Filter size = 3 X 3 -> 3

Input size = 5 X 5 -> 5

Stride = 1X1 -> 1 (1 cell move)

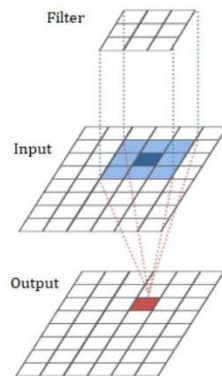
Padding = 0X0 -> 0 (No padding)

output size = (Input size - Filter size + 2 * Padding) * Stride + 1

output size = $(I - F + 2P) * S + 1$

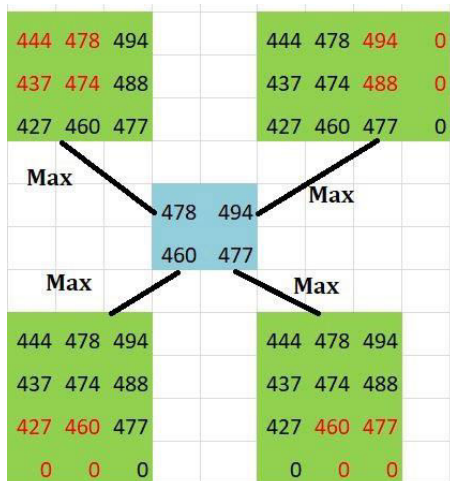
output size = (5 - 3) * 1 + 1

output size = 3 -> 3 X 3



Output image

Pooling Window



Pool size = 2 X 2

Pool type = Max-pool

Stride = 2

Padding = 1

Batch Normalization

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-**batch**. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Regularisation

This technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting. A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X).

$$Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

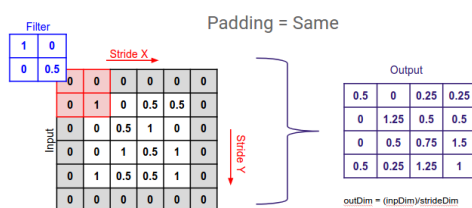
The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

$$RSS = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

Now, this will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

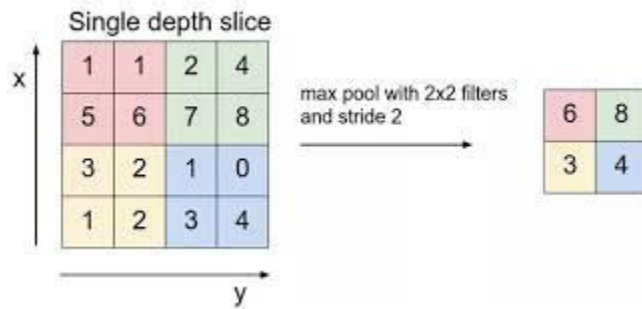
Padding

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a **CNN**. For example, if the **padding** in a **CNN** is set to zero, then every pixel value that is added will be of value zero.



Strides

Stride is the number of pixels shifts over the input matrix. When the **stride** is 1 then we move the filters to 1 pixel at a time. When the **stride** is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a **stride** of 2.



Code:

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

downloaded = drive.CreateFile({'id':"1NwrithRqi1lcpAPadM5Rz0AQRqmr-2DD"})
downloaded.GetContentFile('CatvsDogs.rar')

!unrar x "/content/CatvsDogs.rar" "/content/"

import pandas as pd
import tensorflow as tf
from tensorflow.keras import models, Sequential, layers, preprocessing
import os

file_names=os.listdir("/content/train")
dogorcat=[]
for name in file_names:
    category=name.split('.')[0]
    if category=='dog':
        dogorcat.append("DOG")
    else:
        dogorcat.append("CAT")
train=pd.DataFrame({
    'filename':file_names,
    'category':dogorcat
})

model=models.Sequential()
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(64,64,3)))
#model.add(layers.Conv2D(32,(3,3),activation='relu'))
model.add(layers.BatchNormalization())
```



```

model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.BatchNormalization())
#model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128,(3,3),activation='relu'))
model.add(layers.BatchNormalization())
#model.add(layers.Conv2D(128,(3,3),activation='relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Flatten())
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(256,activation="relu"))
model.add(layers.Dense(128,activation="relu"))
model.add(layers.Dense(2,activation="softmax"))

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

training = preprocessing.image.ImageDataGenerator(rotation_range=15, rescale=1
./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1)
validation=preprocessing.image.ImageDataGenerator(rotation_range=15, rescale=1
./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1)

trainingdata = training.flow_from_dataframe(train,"/content/train",x_col='file
name',y_col='category',target_size=(64,64),class_mode='categorical')

model.fit(trainingdata,epochs=10)

from google.colab import drive
drive.mount('/content/drive')
os.makedirs("drive/My Drive/Models",exist_ok=True)
model.save("drive/My Drive/Models")

```

GUI

```

from PIL import Image,ImageOps
import os
import streamlit as st
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras.preprocessing import image
import numpy as np
st.set_option('deprecation.showfileUploaderEncoding', False)
def load_models(img):
    model = models.load_model('D:/Programs/ML1/ML2/Assignment3/')

```

```

image=img.resize((64,64))
image_array=np.array(image)
#image_array=tf.image.rgb_to_grayscale(image_array)
image_array=(tf.reshape(image_array,(image_array.shape[0],image_array.shap
e[0],3)))/255
image_array=np.array([image_array])

prediction = model.predict_classes(image_array)
return prediction

def upload_images():
    uploaded_file = st.file_uploader("Choose an Image ...", type="jpg")
    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded The image.', use_column_width=True)
        st.write("")
        st.write("Classifying...")
        label = load_models(image)


        if label==0:
            st.write("It is a CAT")
        if label==1:
            st.write("It is a Dog")

if __name__ == "__main__":
    st.header("CAT AND DOG PREDICTION")
    st.write("Upload an image")


    upload_images()

```

Results:

 `model.fit(trainingdata,epochs=10)`

```
Epoch 1/10
782/782 [=====] - 100s 128ms/step - loss: 0.6159 - accuracy: 0.6744
Epoch 2/10
782/782 [=====] - 99s 127ms/step - loss: 0.4912 - accuracy: 0.7582
Epoch 3/10
782/782 [=====] - 101s 129ms/step - loss: 0.4274 - accuracy: 0.7996
Epoch 4/10
782/782 [=====] - 100s 127ms/step - loss: 0.3772 - accuracy: 0.8295
Epoch 5/10
782/782 [=====] - 100s 128ms/step - loss: 0.3511 - accuracy: 0.8447
Epoch 6/10
782/782 [=====] - 100s 128ms/step - loss: 0.3204 - accuracy: 0.8598
Epoch 7/10
782/782 [=====] - 100s 128ms/step - loss: 0.3038 - accuracy: 0.8664
Epoch 8/10
782/782 [=====] - 99s 127ms/step - loss: 0.2918 - accuracy: 0.8721
Epoch 9/10
782/782 [=====] - 99s 127ms/step - loss: 0.2779 - accuracy: 0.8790
Epoch 10/10
782/782 [=====] - 99s 127ms/step - loss: 0.2671 - accuracy: 0.8856
<tensorflow.python.keras.callbacks.History at 0x7f3aaa308710>
```

 `model.fit(trainingdata,validation_data=validationdata,epochs=10)`

```
Epoch 1/10
704/704 [=====] - 93s 131ms/step - loss: 0.6173 - accuracy: 0.6724 - val_loss: 0.5588 - val_accuracy: 0.7252
Epoch 2/10
704/704 [=====] - 92s 130ms/step - loss: 0.4907 - accuracy: 0.7578 - val_loss: 0.5131 - val_accuracy: 0.7348
Epoch 3/10
704/704 [=====] - 91s 129ms/step - loss: 0.4296 - accuracy: 0.7980 - val_loss: 0.3990 - val_accuracy: 0.8232
Epoch 4/10
704/704 [=====] - 91s 129ms/step - loss: 0.3847 - accuracy: 0.8264 - val_loss: 0.3429 - val_accuracy: 0.8432
Epoch 5/10
704/704 [=====] - 90s 128ms/step - loss: 0.3542 - accuracy: 0.8419 - val_loss: 0.3147 - val_accuracy: 0.8576
Epoch 6/10
704/704 [=====] - 89s 127ms/step - loss: 0.3309 - accuracy: 0.8520 - val_loss: 0.3279 - val_accuracy: 0.8532
Epoch 7/10
704/704 [=====] - 90s 127ms/step - loss: 0.3190 - accuracy: 0.8610 - val_loss: 0.3466 - val_accuracy: 0.8492
Epoch 8/10
704/704 [=====] - 90s 128ms/step - loss: 0.3018 - accuracy: 0.8662 - val_loss: 0.3381 - val_accuracy: 0.8596
Epoch 9/10
704/704 [=====] - 93s 131ms/step - loss: 0.2969 - accuracy: 0.8715 - val_loss: 0.3218 - val_accuracy: 0.8596
Epoch 10/10
704/704 [=====] - 92s 131ms/step - loss: 0.2792 - accuracy: 0.8791 - val_loss: 0.2868 - val_accuracy: 0.8824
<tensorflow.python.keras.callbacks.History at 0x7f89db6da6a0>
```

GUI

CAT AND DOG PREDICTION

Upload an image

Choose an Image ...

_111434468_gettyimages-1143489763.jpg

[browse files](#)



Uploaded The image.

Classifying...

It is a CAT

Conclusion:

Thus, we have understood how and where CNN is used and how it is programmed in TensorFlow.

We have also been able to polish GUI creation skills even further.

Assignment No. 4

Aim: Develop face recognition system using CNN. Create a dataset of minimum 20 students from your class. Check and validate the accuracy of the model.

☑ Apply dimensionality reduction on input image and plot the change in accuracy of system.

Objectives:

1. To learn Data set creation
2. To learn data normalization

Theory:

Dataset creation steps

1. Articulate the problem early.
2. Establish data collection mechanisms.
3. Format data to make it consistent.
4. Reduce data.
5. Complete data cleaning.
6. Decompose data.
7. Rescale data.
8. Discretize data.

Image Augmentation

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. Image data augmentation is used to expand the training dataset in order to improve the performance and ability of the model to generalize.

The intent is to expand the training dataset with new, plausible examples. This means, variations of the training set image that are likely to be seen by the model. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right. A vertical flip of the photo of a cat does not make sense and would probably not be appropriate given that the model is very unlikely to see a photo of an upside-down cat.

Libraries for image augmentation

There are a lot of image augmentations packages

- skimage
- opencv
- imgaug
- Albumentations
- Augmentor

- Keras(ImageDataGenerator class)

Fit_generator, validate_generator, predict_generator

fit_generator

```
fit_generator(  
    generator, steps_per_epoch=None, epochs=1, verbose=1, callbacks=None,  
    validation_data=None, validation_steps=None, validation_freq=1,  
    class_weight=None, max_queue_size=10, workers=1, use_multiprocessing=False,  
    shuffle=True, initial_epoch=0  
)
```

Fits the model on data yielded batch-by-batch by a Python generator.

predict_generator

```
predict_generator(  
    generator, steps=None, callbacks=None, max_queue_size=10, workers=1,  
    use_multiprocessing=False, verbose=0  
)
```

Generates predictions for the input samples from a data generator.

evaluate_generator

```
evaluate_generator(  
    generator, steps=None, callbacks=None, max_queue_size=10, workers=1,  
    use_multiprocessing=False, verbose=0  
)
```

Evaluates the model on a data generator.

Code:

MTCNN

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from matplotlib.patches import Circle
from mtcnn.mtcnn import MTCNN
from PIL import Image
from numpy import asarray
def draw_faces(filename, result_list):
    data = plt.imread(filename)
    for i in range(len(result_list)):
        # get coordinates
        x1, y1, width, height = result_list[i]['box']
        x2, y2 = x1 + width, y1 + height
        if x1 < 0:
            x1 = 0
        if x2 < 0:
            x2 = 0
        if y1 < 0:
            y1 = 0
        if y2 < 0:
            y2 = 0
        plt.subplot(1, len(result_list), i+1)
        plt.axis('off')
        plt.imshow(data[y1:y2, x1:x2])
        cv2.imwrite(filename, data[y1:y2, x1:x2])
    plt.show()
```

```
import glob
import cv2
path = glob.glob("FaceRecog/*.jpg")
cv_img = []
for img in path:
    filename = img
    image = Image.open(filename)
    image = image.convert('RGB')
    pixels = asarray(image)
    detector = MTCNN()
    faces = detector.detect_faces(pixels)
    draw_faces(filename, faces)
```

MODEL

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras import models, Sequential, layers, preprocessing
import keras
import os
import mtcnn

file_names=os.listdir("FaceRecog")
NameArray=[]
for name in file_names:
    category=name.split('.')[0]
    if category=='gourishankar':
        NameArray.append('Gourishankar')
    elif category=='Aditya_Panchwagh':
        NameArray.append("Aditya")
    elif category=="Dhananjay_Jha":
        NameArray.append("Dhananjay")
    elif category=='Habil_Bhagat':
        NameArray.append("Habil")
    elif category=="Karan_Mahajan":
        NameArray.append("Karan")
    elif category=='Kartik_Jawanjal':
        NameArray.append("Kartik")
    elif category=="Krish_Shah":
        NameArray.append("Krish")
    elif category=='Manas_Oswal':
        NameArray.append("Manas")
    elif category=="Mayank_Modi":
        NameArray.append("Mayank")
    elif category=='Shubham_Pagare':
        NameArray.append("Shubham")
    elif category=="Vishal_Kasa":
        NameArray.append("Kasa")

train=pd.DataFrame({
    'filename':file_names,
    'category':NameArray
})

from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator,load_img
train_df,validate_df = train_test_split(train,test_size=0.2, random_state=0)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
training = preprocessing.image.ImageDataGenerator(rotation_range=5, rescale=1./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1)
```



```

trainingdata = training.flow_from_dataframe(train_df, "FaceRecog", x_col='filename', y_col='category', target_size=(224,224), class_mode='categorical')
validation = ImageDataGenerator(rotation_range=5, rescale=1./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1)
validationdata = validation.flow_from_dataframe(validate_df, "FaceRecog", x_col='filename', y_col='category', target_size=(224,224), class_mode='categorical')

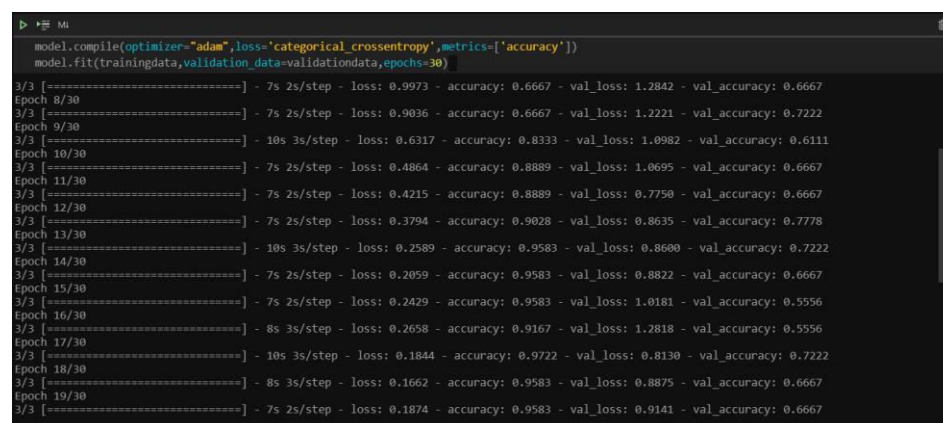
from tensorflow.keras.applications.vgg16 import VGG16
base = VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))
base.trainable = False
model = models.Sequential()
model.add(base)
model.add(layers.Flatten())
model.add(layers.Dense(400, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainingdata, validation_data=validationdata, epochs=30)

_, validation_acc = model.evaluate(validationdata, verbose=0)
print(validation_acc)

```

Results:



```

model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(trainingdata, validation_data=validationdata, epochs=30)
3/3 [=====] - 7s 2s/step - loss: 0.9973 - accuracy: 0.6667 - val_loss: 1.2842 - val_accuracy: 0.6667
Epoch 8/30
3/3 [=====] - 7s 2s/step - loss: 0.9036 - accuracy: 0.6667 - val_loss: 1.2221 - val_accuracy: 0.7222
Epoch 9/30
3/3 [=====] - 10s 3s/step - loss: 0.6317 - accuracy: 0.8333 - val_loss: 1.0982 - val_accuracy: 0.6111
Epoch 10/30
3/3 [=====] - 7s 2s/step - loss: 0.4864 - accuracy: 0.8889 - val_loss: 1.0695 - val_accuracy: 0.6667
Epoch 11/30
3/3 [=====] - 7s 2s/step - loss: 0.4215 - accuracy: 0.8889 - val_loss: 0.7750 - val_accuracy: 0.6667
Epoch 12/30
3/3 [=====] - 7s 2s/step - loss: 0.3794 - accuracy: 0.9028 - val_loss: 0.8635 - val_accuracy: 0.7778
Epoch 13/30
3/3 [=====] - 10s 3s/step - loss: 0.2589 - accuracy: 0.9583 - val_loss: 0.8600 - val_accuracy: 0.7222
Epoch 14/30
3/3 [=====] - 7s 2s/step - loss: 0.2059 - accuracy: 0.9583 - val_loss: 0.8822 - val_accuracy: 0.6667
Epoch 15/30
3/3 [=====] - 7s 2s/step - loss: 0.2429 - accuracy: 0.9583 - val_loss: 1.0181 - val_accuracy: 0.5556
Epoch 16/30
3/3 [=====] - 8s 3s/step - loss: 0.2658 - accuracy: 0.9167 - val_loss: 1.2818 - val_accuracy: 0.5556
Epoch 17/30
3/3 [=====] - 10s 3s/step - loss: 0.1844 - accuracy: 0.9722 - val_loss: 0.8130 - val_accuracy: 0.7222
Epoch 18/30
3/3 [=====] - 8s 3s/step - loss: 0.1662 - accuracy: 0.9583 - val_loss: 0.8875 - val_accuracy: 0.6667
Epoch 19/30
3/3 [=====] - 7s 2s/step - loss: 0.1874 - accuracy: 0.9583 - val_loss: 0.9141 - val_accuracy: 0.6667

```

```

3/3 [=====] - 7s 2s/step - loss: 0.1874 - accuracy: 0.9583 - val_loss: 0.9141 - val_accuracy: 0.6667
Epoch 20/30
3/3 [=====] - 10s 3s/step - loss: 0.1647 - accuracy: 0.9861 - val_loss: 0.7870 - val_accuracy: 0.7222
Epoch 21/30
3/3 [=====] - 8s 3s/step - loss: 0.1514 - accuracy: 0.9583 - val_loss: 0.7556 - val_accuracy: 0.7222
Epoch 22/30
3/3 [=====] - 7s 2s/step - loss: 0.1013 - accuracy: 0.9861 - val_loss: 0.7539 - val_accuracy: 0.6667
Epoch 23/30
3/3 [=====] - 7s 2s/step - loss: 0.1262 - accuracy: 0.9861 - val_loss: 0.6245 - val_accuracy: 0.7222
Epoch 24/30
3/3 [=====] - 8s 3s/step - loss: 0.1059 - accuracy: 0.9861 - val_loss: 0.7109 - val_accuracy: 0.6111
Epoch 25/30
3/3 [=====] - 11s 4s/step - loss: 0.0766 - accuracy: 0.9861 - val_loss: 0.9960 - val_accuracy: 0.7222
Epoch 26/30
3/3 [=====] - 11s 4s/step - loss: 0.0996 - accuracy: 0.9583 - val_loss: 0.9721 - val_accuracy: 0.6111
Epoch 27/30
3/3 [=====] - 8s 3s/step - loss: 0.0942 - accuracy: 1.0000 - val_loss: 0.9337 - val_accuracy: 0.7222
Epoch 28/30
3/3 [=====] - 7s 2s/step - loss: 0.0698 - accuracy: 0.9861 - val_loss: 0.7403 - val_accuracy: 0.6667
Epoch 29/30
3/3 [=====] - 11s 4s/step - loss: 0.0663 - accuracy: 1.0000 - val_loss: 0.6007 - val_accuracy: 0.7778
Epoch 30/30
3/3 [=====] - 8s 3s/step - loss: 0.0881 - accuracy: 0.9861 - val_loss: 0.7000 - val_accuracy: 0.6667
tensorflow.python.keras.callbacks.History at 0x14c6e916398

```

```

▶ ▶ M
_, validation_acc = model.evaluate(validationdata, verbose=0)
print(validation_acc)

0.7777777910232544

```

Conclusion:

Thus, we have understood how to create Face recognition system is used and how it is programmed in TensorFlow.

Plus we have also learned to use various face detection algorithms

Assignment No. 5

Aim: Write a program to demonstrate the change in accuracy/loss/convergence time with change in optimizers like stochastic gradient descent, adam, adagrad, RMSprop and Nadam for any suitable application

Objectives:

1. To learn optimization algorithms
2. To learn and understand hyperparameters

Theory:

SGD, Adam, RMSprop, Nadam

The word '*stochastic*' means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.

Adam is a replacement optimization algorithm for stochastic gradient descent for training **deep learning** models. **Adam** combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9.

Nadam combines NAG and Adam. Nadam is employed for noisy gradients or for gradients with high curvatures. The learning process is accelerated by summing up the exponential decay of the moving averages for the previous and current gradient

Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
from tensorflow.keras.applications.vgg16 import VGG16
```

```

(train_images,train_labels),(test_images,test_labels)=datasets.cifar100.load_data()
train_images=train_images/255
test_images=test_images/255

base=VGG16(include_top=False,input_shape=(32,32,3))
base.trainable=False
model=models.Sequential()
model.add(layers.Flatten())
model.add(layers.Dense(1200,activation="relu"))
model.add(layers.Dense(100,activation="softmax"))
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=200)

model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

model.compile(optimizer="adagrad",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

model.compile(optimizer="rmsprop",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

model.compile(optimizer="nadam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

```

Results:

```

▶ MI
model.compile(optimizer="rmsprop",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

Epoch 1/10
50/50 [=====] - 8s 165ms/step - loss: 3.2918 - accuracy: 0.2295 - val_loss: 3.4460 - val_accuracy: 0.2053
Epoch 2/10
50/50 [=====] - 8s 163ms/step - loss: 3.1533 - accuracy: 0.2452 - val_loss: 3.3486 - val_accuracy: 0.2210
Epoch 3/10
50/50 [=====] - 8s 159ms/step - loss: 3.1276 - accuracy: 0.2473 - val_loss: 3.3681 - val_accuracy: 0.2141
Epoch 4/10
50/50 [=====] - 8s 163ms/step - loss: 3.1184 - accuracy: 0.2512 - val_loss: 3.3351 - val_accuracy: 0.2253
Epoch 5/10
50/50 [=====] - 9s 174ms/step - loss: 3.1034 - accuracy: 0.2523 - val_loss: 3.4193 - val_accuracy: 0.2031
Epoch 6/10
50/50 [=====] - 8s 167ms/step - loss: 3.0861 - accuracy: 0.2577 - val_loss: 3.3914 - val_accuracy: 0.2155
Epoch 7/10
50/50 [=====] - 8s 152ms/step - loss: 3.0775 - accuracy: 0.2616 - val_loss: 3.3460 - val_accuracy: 0.2236
Epoch 8/10
50/50 [=====] - 8s 158ms/step - loss: 3.0583 - accuracy: 0.2630 - val_loss: 3.3535 - val_accuracy: 0.2227
Epoch 9/10
50/50 [=====] - 8s 156ms/step - loss: 3.0469 - accuracy: 0.2653 - val_loss: 3.3743 - val_accuracy: 0.2155
Epoch 10/10
50/50 [=====] - 7s 149ms/step - loss: 3.0225 - accuracy: 0.2699 - val_loss: 3.4018 - val_accuracy: 0.2176

<tensorflow.python.keras.callbacks.History at 0x1490ffe588>

```

```

▶ MI
model.compile(optimizer="adagrad",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

Epoch 1/10
50/50 [=====] - 6s 127ms/step - loss: 2.9720 - accuracy: 0.2847 - val_loss: 3.2421 - val_accuracy: 0.2382
Epoch 2/10
50/50 [=====] - 7s 132ms/step - loss: 2.9712 - accuracy: 0.2851 - val_loss: 3.2420 - val_accuracy: 0.2389
Epoch 3/10
50/50 [=====] - 6s 128ms/step - loss: 2.9706 - accuracy: 0.2854 - val_loss: 3.2418 - val_accuracy: 0.2390
Epoch 4/10
50/50 [=====] - 6s 123ms/step - loss: 2.9700 - accuracy: 0.2852 - val_loss: 3.2417 - val_accuracy: 0.2383
Epoch 5/10
50/50 [=====] - 7s 131ms/step - loss: 2.9695 - accuracy: 0.2851 - val_loss: 3.2415 - val_accuracy: 0.2386
Epoch 6/10
50/50 [=====] - 7s 133ms/step - loss: 2.9690 - accuracy: 0.2853 - val_loss: 3.2413 - val_accuracy: 0.2384
Epoch 7/10
50/50 [=====] - 7s 134ms/step - loss: 2.9685 - accuracy: 0.2850 - val_loss: 3.2411 - val_accuracy: 0.2385
Epoch 8/10
50/50 [=====] - 6s 124ms/step - loss: 2.9680 - accuracy: 0.2857 - val_loss: 3.2410 - val_accuracy: 0.2376
Epoch 9/10
50/50 [=====] - 7s 134ms/step - loss: 2.9675 - accuracy: 0.2855 - val_loss: 3.2408 - val_accuracy: 0.2381
Epoch 10/10
50/50 [=====] - 6s 125ms/step - loss: 2.9670 - accuracy: 0.2855 - val_loss: 3.2407 - val_accuracy: 0.2387

<tensorflow.python.keras.callbacks.History at 0x14900f03c8>

```

```

▶ MI
model.compile(optimizer="sgd",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

Epoch 1/10
50/50 [=====] - 7s 136ms/step - loss: 3.0500 - accuracy: 0.2682 - val_loss: 3.2709 - val_accuracy: 0.2341
Epoch 2/10
50/50 [=====] - 6s 114ms/step - loss: 3.0152 - accuracy: 0.2766 - val_loss: 3.2599 - val_accuracy: 0.2336
Epoch 3/10
50/50 [=====] - 6s 112ms/step - loss: 3.0031 - accuracy: 0.2784 - val_loss: 3.2541 - val_accuracy: 0.2349
Epoch 4/10
50/50 [=====] - 5s 110ms/step - loss: 2.9955 - accuracy: 0.2802 - val_loss: 3.2503 - val_accuracy: 0.2384
Epoch 5/10
50/50 [=====] - 6s 116ms/step - loss: 2.9901 - accuracy: 0.2808 - val_loss: 3.2479 - val_accuracy: 0.2389
Epoch 6/10
50/50 [=====] - 6s 125ms/step - loss: 2.9859 - accuracy: 0.2823 - val_loss: 3.2465 - val_accuracy: 0.2368
Epoch 7/10
50/50 [=====] - 6s 118ms/step - loss: 2.9828 - accuracy: 0.2820 - val_loss: 3.2453 - val_accuracy: 0.2391
Epoch 8/10
50/50 [=====] - 6s 117ms/step - loss: 2.9800 - accuracy: 0.2834 - val_loss: 3.2443 - val_accuracy: 0.2390
Epoch 9/10
50/50 [=====] - 6s 116ms/step - loss: 2.9776 - accuracy: 0.2835 - val_loss: 3.2436 - val_accuracy: 0.2372
Epoch 10/10
50/50 [=====] - 6s 114ms/step - loss: 2.9756 - accuracy: 0.2843 - val_loss: 3.2427 - val_accuracy: 0.2385

<tensorflow.python.keras.callbacks.History at 0x1496eb90b08>

```

```
model.compile(optimizer="adam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=200)

Epoch 1/10
200/200 [=====] - 22s 109ms/step - loss: 4.2006 - accuracy: 0.0757 - val_loss: 3.8973 - val_accuracy: 0.1107
Epoch 2/10
200/200 [=====] - 23s 117ms/step - loss: 3.7537 - accuracy: 0.1343 - val_loss: 3.7244 - val_accuracy: 0.1415
Epoch 3/10
200/200 [=====] - 21s 107ms/step - loss: 3.6068 - accuracy: 0.1607 - val_loss: 3.6081 - val_accuracy: 0.1604
Epoch 4/10
200/200 [=====] - 22s 111ms/step - loss: 3.5108 - accuracy: 0.1801 - val_loss: 3.5507 - val_accuracy: 0.1779
Epoch 5/10
200/200 [=====] - 23s 113ms/step - loss: 3.4321 - accuracy: 0.1927 - val_loss: 3.5164 - val_accuracy: 0.1831
Epoch 6/10
200/200 [=====] - 23s 113ms/step - loss: 3.3603 - accuracy: 0.2066 - val_loss: 3.4489 - val_accuracy: 0.1994
Epoch 7/10
200/200 [=====] - 21s 107ms/step - loss: 3.3046 - accuracy: 0.2150 - val_loss: 3.4260 - val_accuracy: 0.1993
Epoch 8/10
200/200 [=====] - 20s 102ms/step - loss: 3.2435 - accuracy: 0.2288 - val_loss: 3.4075 - val_accuracy: 0.2041
Epoch 9/10
200/200 [=====] - 18s 89ms/step - loss: 3.2154 - accuracy: 0.2325 - val_loss: 3.3717 - val_accuracy: 0.2109
Epoch 10/10
200/200 [=====] - 14s 68ms/step - loss: 3.1621 - accuracy: 0.2409 - val_loss: 3.3724 - val_accuracy: 0.2089

<tensorflow.python.keras.callbacks.History at 0x1496e5bd848>

model.compile(optimizer="nadam",loss="sparse_categorical_crossentropy",metrics=["accuracy"])
model.fit(train_images,train_labels,epochs=10,validation_data=(test_images,test_labels),steps_per_epoch=50)

Epoch 1/10
50/50 [=====] - 10s 193ms/step - loss: 2.9010 - accuracy: 0.2938 - val_loss: 3.3019 - val_accuracy: 0.2345
Epoch 2/10
50/50 [=====] - 9s 181ms/step - loss: 2.8911 - accuracy: 0.2963 - val_loss: 3.3082 - val_accuracy: 0.2314
Epoch 3/10
50/50 [=====] - 9s 182ms/step - loss: 2.8688 - accuracy: 0.3027 - val_loss: 3.2694 - val_accuracy: 0.2442
Epoch 4/10
50/50 [=====] - 9s 188ms/step - loss: 2.8464 - accuracy: 0.3037 - val_loss: 3.2894 - val_accuracy: 0.2374
Epoch 5/10
50/50 [=====] - 10s 200ms/step - loss: 2.8303 - accuracy: 0.3091 - val_loss: 3.2336 - val_accuracy: 0.2472
Epoch 6/10
50/50 [=====] - 10s 193ms/step - loss: 2.8139 - accuracy: 0.3141 - val_loss: 3.2741 - val_accuracy: 0.2391
Epoch 7/10
50/50 [=====] - 10s 192ms/step - loss: 2.8008 - accuracy: 0.3150 - val_loss: 3.2407 - val_accuracy: 0.2472
Epoch 8/10
50/50 [=====] - 10s 194ms/step - loss: 2.7790 - accuracy: 0.3185 - val_loss: 3.3214 - val_accuracy: 0.2366
Epoch 9/10
50/50 [=====] - 10s 196ms/step - loss: 2.7694 - accuracy: 0.3205 - val_loss: 3.2575 - val_accuracy: 0.2456
Epoch 10/10
50/50 [=====] - 10s 197ms/step - loss: 2.7341 - accuracy: 0.3288 - val_loss: 3.2599 - val_accuracy: 0.2506

<tensorflow.python.keras.callbacks.History at 0x149002eb108>
```

Conclusion:

Thus, we have understood the difference in performance of various optimisation algorithms

Assignment No. 6

Aim: Apply transfer learning with pre-trained VGG16 model on assignment 3 and analyze the result.

Objectives:

1. To learn pre-trained models
2. To learn transfer learning

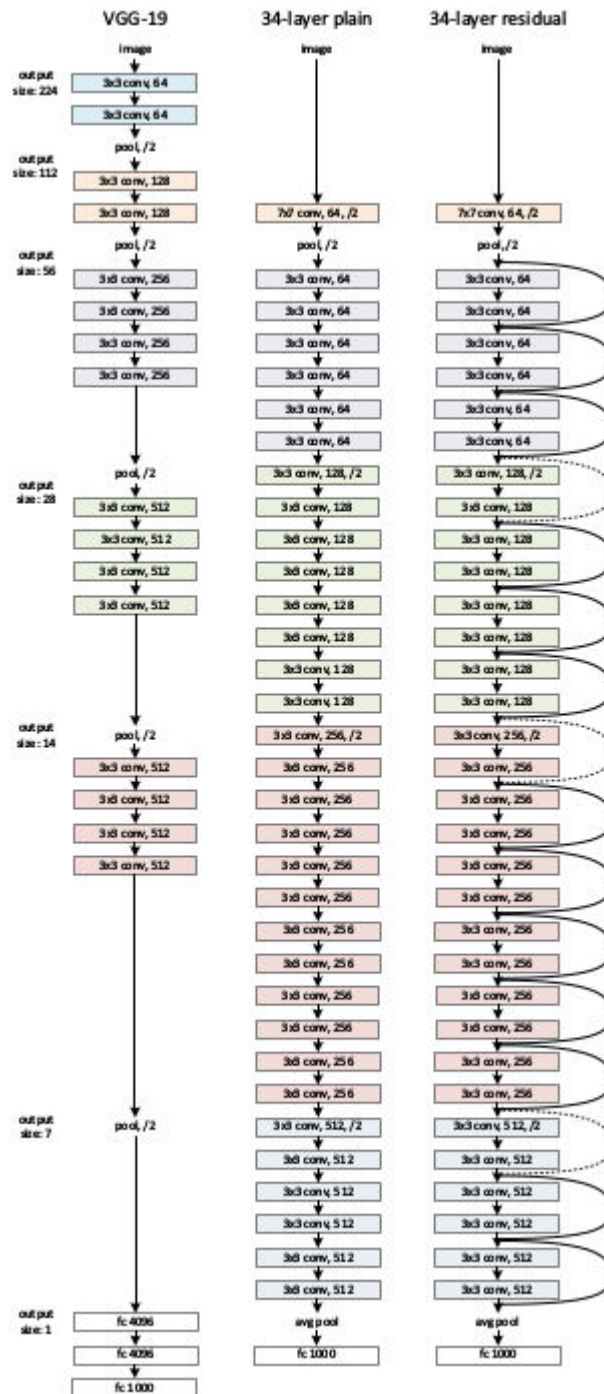
Theory:

Transfer learning

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This area of research bears some relation to the long history of psychological literature on transfer of learning, although formal ties between the two fields are limited. From the practical standpoint, reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency of a reinforcement learning agent.

ResNet 50

1. Use 3*3 filters mostly.
2. Down sampling with CNN layers with stride 2.
3. Global average pooling layer and a 1000-way fully-connected layer with Softmax in the end.



Plain VGG and VGG with Residual Blocks

There are two kinds of residual connections:

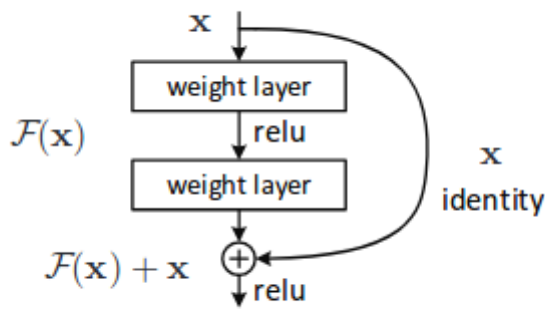


Figure 2. Residual learning: a building block.

Residual block

1. The identity shortcuts (x) can be directly used when the input and output are of the same dimensions.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \quad (1)$$

Residual block function when input and output dimensions are same

2. When the dimensions change, A) The shortcut still performs identity mapping, with extra zero entries padded with the increased dimension. B) The projection shortcut is used to match the dimension (done by 1×1 conv) using the following formula

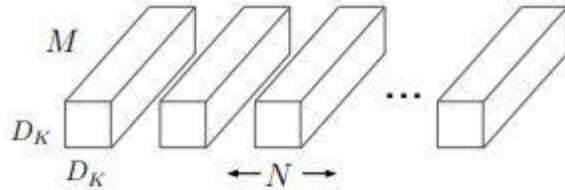
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s \mathbf{x}. \quad (2)$$

Residual block function when the input and output dimensions are not same.

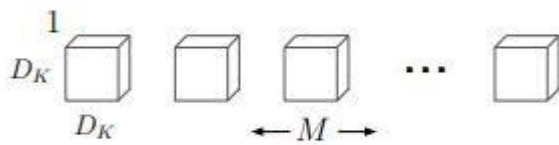
The first case adds no extra parameters, the second one adds in the form of $W_{\{s\}}$

MOBILENET

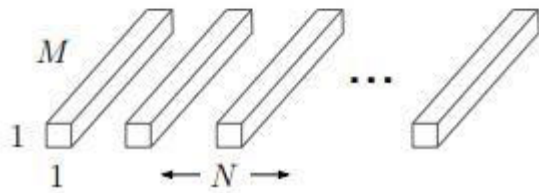
MobileNet is an efficient and portable CNN architecture that is used in real world applications. MobileNets primarily use **depthwise seperable convolutions** in place of the standard convolutions used in earlier architectures to build lighter models. MobileNets introduce two new global hyperparameters (width multiplier and resolution multiplier) that allow model developers to trade off **latency** or **accuracy** for speed and low size depending on their requirements.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



Architecture

MobileNets are built on depthwise seperable convolution layers. Each depthwise seperable convolution layer consists of a depthwise convolution and a pointwise convolution. Counting depthwise and pointwise convolutions as separate layers, a MobileNet has 28 layers. A standard MobileNet has 4.2 million parameters which can be further reduced by tuning the width multiplier hyperparameter appropriately.

The size of the input image is $224 \times 224 \times 3$.

The detailed architecture of a MobileNet is given below :

TYPE	STRIDE	KERNEL SHAPE	INPUT SIZE
Conv	2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw	1	$3 \times 3 \times 32$	$112 \times 112 \times 32$
Conv	1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw	2	$3 \times 3 \times 64$	$112 \times 112 \times 64$
Conv	1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$

TYPE	STRIDE	KERNEL SHAPE	INPUT SIZE
Conv dw	1	$3 \times 3 \times 128$	$56 \times 56 \times 128$
Conv	1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw	2	$3 \times 3 \times 128$	$56 \times 56 \times 128$
Conv	1	$1 \times 1 \times 128 \times 256$	$56 \times 56 \times 128$
Conv dw	1	$3 \times 3 \times 256$	$28 \times 28 \times 256$
Conv	1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw	2	$3 \times 3 \times 256$	$28 \times 28 \times 256$
Conv	1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
Conv dw	1	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw	1	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw	1	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw	1	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw	1	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw	1	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw	2	$3 \times 3 \times 512$	$14 \times 14 \times 512$
Conv	1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw	2	$3 \times 3 \times 1024$	$7 \times 7 \times 1024$
Conv	1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool	1	Pool 7×7	$7 \times 7 \times 1024$
Fully Connected	1	1024×1000	$1 \times 1 \times 1024$
Softmax	1	Classifier	$1 \times 1 \times 1000$

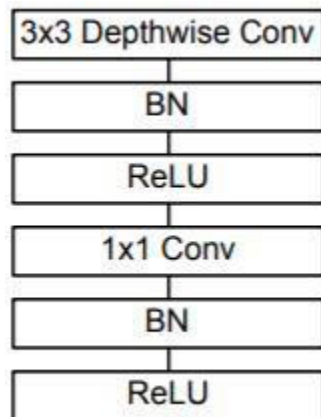
Standard Convolution layer :

A single standard convolution unit (denoted by **Conv** in the table above) looks like this :



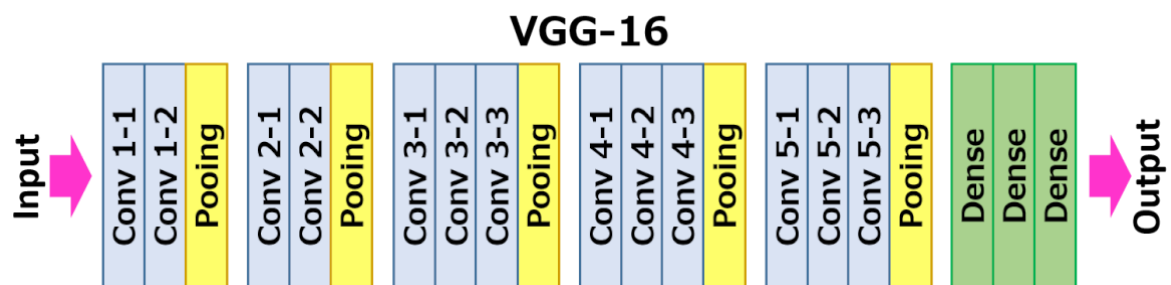
Depthwise seperable Convolution layer

A single Depthwise seperable convolution unit (denoted by **Conv dw** in the table above) looks like this :

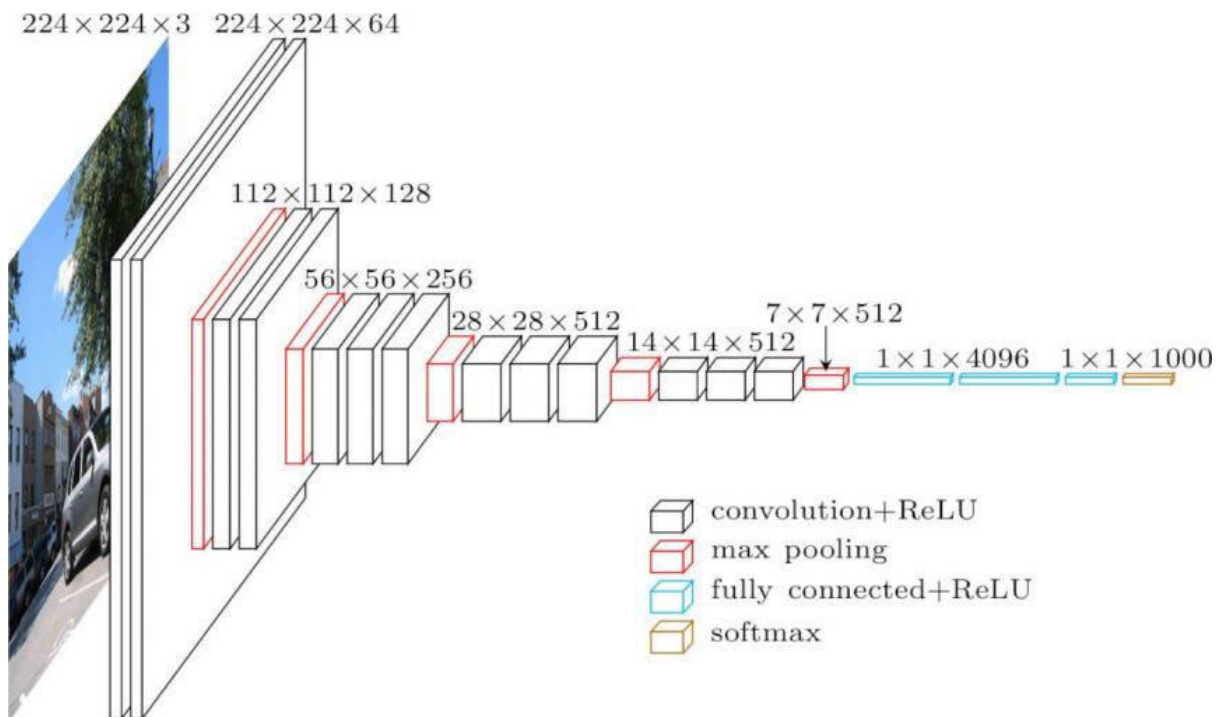


VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to [ILSVRC-2014](#). It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3x3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.



The architecture depicted below is VGG16.



VGG16 Architecture

The input to conv1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3x3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations, it also utilizes 1x1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3x3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2x2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the softmax layer. The configuration of the fully connected layers is the same in all networks.

Code:

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras import models, Sequential, layers, preprocessing
import os
from tensorflow.keras.applications.vgg16 import VGG16

file_names=os.listdir("/content/train")
dogorcat=[]
for name in file_names:
```

```

        category=name.split('.')[0]
        if category=='dog':
            dogorcat.append("DOG")
        else:
            dogorcat.append("CAT")
train=pd.DataFrame({
    'filename':file_names,
    'category':dogorcat
})

base=VGG16(include_top=False,input_shape=(224,224,3),weights='imagenet')
base.trainable=False
model=models.Sequential()
model.add(base)
model.add(layers.Flatten())
model.add(layers.Dense(120,activation="relu"))
model.add(layers.Dense(2,activation="softmax"))

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

from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator,load_img
train_df,validate_df = train_test_split(train,test_size=0.1, random_state=42)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
training = preprocessing.image.ImageDataGenerator(rotation_range=15, rescale=1
./255, shear_range=0.1, zoom_range=0.2, horizontal_flip=True, width_shift_rang
e=0.1, height_shift_range=0.1)
total_train=train_df.shape[0]
total_validate=validate_df.shape[0]

trainingdata = training.flow_from_dataframe(train_df,"/content/train",x_col='f
ilename',y_col='category',target_size=(224,224),class_mode='categorical',batch
_size=4)
validation = ImageDataGenerator(rescale=1./255)
validationdata = validation.flow_from_dataframe(validate_df,"/content/train",
x_col='filename',y_col='category',target_size=(224,224),class_mode='categorica
l',batch_size=4)

model.fit(trainingdata,validation_data=validationdata,epochs=10,steps_per_epoc
h=total_train//200,validation_steps=total_validate//200)

model.evaluate(validationdata)

```

Results:

```
model.fit(trainingdata,validation_data=validationdata,epochs=10,steps_per_epoch=total_train//200,validation_steps=total_validate//200)

Epoch 1/10
112/112 [=====] - 256s 2s/step - loss: 0.7865 - accuracy: 0.7366 - val_loss: 0.8776 - val_accuracy: 0.6667
Epoch 2/10
112/112 [=====] - 257s 2s/step - loss: 0.5440 - accuracy: 0.7723 - val_loss: 0.1737 - val_accuracy: 0.9167
Epoch 3/10
112/112 [=====] - 255s 2s/step - loss: 0.4249 - accuracy: 0.8393 - val_loss: 0.5555 - val_accuracy: 0.8125
Epoch 4/10
112/112 [=====] - 255s 2s/step - loss: 0.3796 - accuracy: 0.8549 - val_loss: 0.3619 - val_accuracy: 0.8125
Epoch 5/10
112/112 [=====] - 252s 2s/step - loss: 0.3398 - accuracy: 0.8527 - val_loss: 0.2243 - val_accuracy: 0.9375
Epoch 6/10
112/112 [=====] - 254s 2s/step - loss: 0.3460 - accuracy: 0.8482 - val_loss: 0.1937 - val_accuracy: 0.9167
Epoch 7/10
112/112 [=====] - 254s 2s/step - loss: 0.3037 - accuracy: 0.8661 - val_loss: 0.2235 - val_accuracy: 0.8750
Epoch 8/10
112/112 [=====] - 255s 2s/step - loss: 0.2772 - accuracy: 0.8884 - val_loss: 0.2347 - val_accuracy: 0.8958
Epoch 9/10
112/112 [=====] - 255s 2s/step - loss: 0.3160 - accuracy: 0.8728 - val_loss: 0.2730 - val_accuracy: 0.8958
Epoch 10/10
112/112 [=====] - 253s 2s/step - loss: 0.3103 - accuracy: 0.8728 - val_loss: 0.4754 - val_accuracy: 0.7917
<tensorflow.python.keras.callbacks.History at 0x7fcd9d05cbe0>

[ ] model.evaluate(validationdata)

625/625 [=====] - 1250s 2s/step - loss: 0.3219 - accuracy: 0.8564
[0.3218751549720764, 0.8564000129699707]
```

Conclusion:

Thus, we have understood the importance of Transfer learning and also the usage of transfer learning in tensorflow

Assignment No. 7

Aim: Develop RNN model for Cryptocurrency pricing prediction or text sentiment analysis

Objectives:

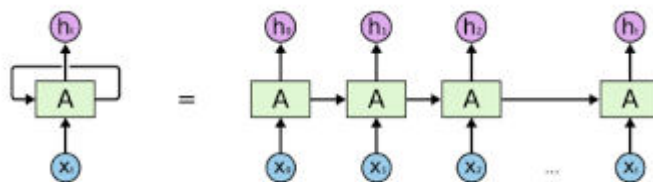
1. To learn RNN
2. To learn and implement LSTM

Theory:

RNN

A **recurrent neural network (RNN)** is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition^[4] or speech recognition.

The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.



An unrolled recurrent neural network.

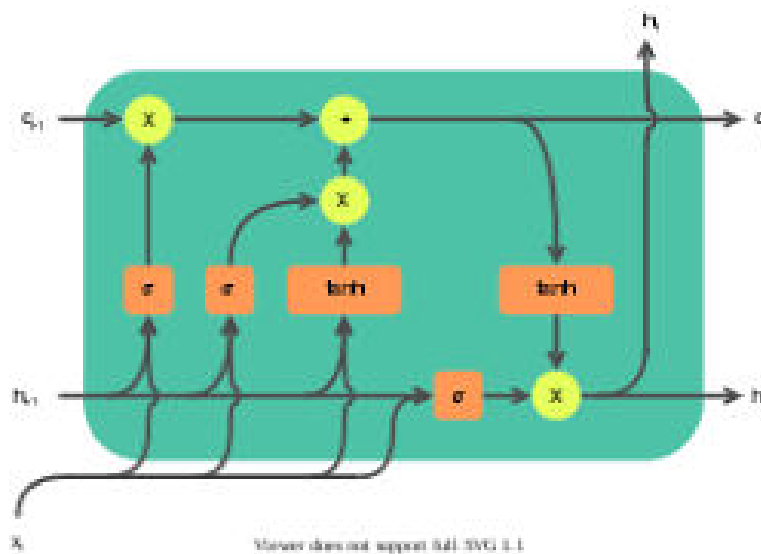
LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

A common LSTM unit is composed of a **cell**, an **input gate**, an **output gate** and a **forget gate**. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.



BPTT

Backpropagation refers to two things:

- The mathematical method used to calculate derivatives and an application of the derivative chain rule.
- The training algorithm for updating network weights to minimize error.

It is this latter understanding of backpropagation that we are using here.

The goal of the backpropagation training algorithm is to modify the weights of a neural network in order to minimize the error of the network outputs compared to some expected output in response to corresponding inputs.

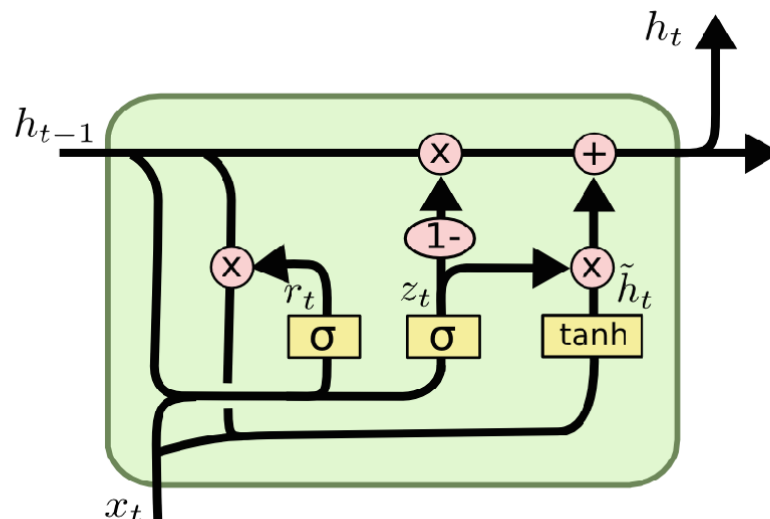
It is a supervised learning algorithm that allows the network to be corrected with regard to the specific errors made.

The general algorithm is as follows:

1. Present a training input pattern and propagate it through the network to get an output.
2. Compare the predicted outputs to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Adjust the weights to minimize the error.
5. Repeat.

GRU

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate. GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling and natural language processing was found to be similar to that of LSTM. GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets.



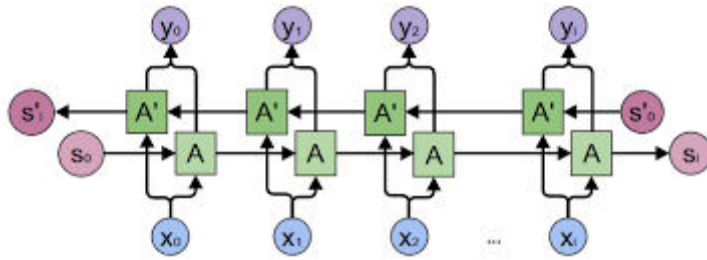
Bi directional RNN

Bidirectional recurrent neural networks(RNN) are really just putting two independent RNNs together. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are usually concatenated at each time step, though there are other options, e.g. summation.

This structure allows the networks to have both backward and forward information about the sequence at every time step. The concept seems easy enough. But when it comes to actually implementing a neural network which utilizes bidirectional structure, confusion arises...

The first confusion is about **the way to forward the outputs of a bidirectional RNN to a dense neural network**.

The second confusion is about the **returned hidden states**. In seq2seq models, we'll want hidden states from the encoder to initialize the hidden states of the decoder. Intuitively, if we can only choose hidden states at one time step(as in PyTorch), we'd want the one at which the RNN just consumed the last input in the sequence. But **if** the hidden states of time step n (the last one) are returned, as before, we'll have the hidden states of the reversed RNN with only one step of inputs seen.



Code:

```
import os
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import Bidirectional, Dropout, Activation, Dense, LSTM
from tensorflow.keras.models import Sequential, Model

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)
df = pd.read_csv("BTC-USD.csv", parse_dates=['Date'])
df = df.sort_values('Date')
scaler = MinMaxScaler()
close_price = df.Close.values.reshape(-1, 1)
scaled_close = scaler.fit_transform(close_price)
SEQ_LEN = 100

def to_sequences(data, seq_len):
    d = []
    for index in range(len(data) - seq_len):
        d.append(data[index: index + seq_len])

    return np.array(d)

def preprocess(data_raw, seq_len, train_split):
    data = to_sequences(data_raw, seq_len)
    num_train = int(train_split * data.shape[0])
    X_train = data[:num_train, :-1, :]
```

```

y_train = data[:num_train, -1, :]
X_test = data[num_train:, :-1, :]
y_test = data[num_train:, -1, :]

return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = preprocess(scaled_close, SEQ_LEN, train_split = 0.85)
DROPOUT = 0.2
WINDOW_SIZE = SEQ_LEN - 1

model = keras.Sequential()
model.add(LSTM(input_shape=((WINDOW_SIZE, X_train.shape[-1])), units=200))
model.add(Dropout(rate=DROPOUT))
#model.add(LSTM(units=100))
model.add(Dense(units=100))
model.add(Dense(1, activation="sigmoid"))
BATCH_SIZE = 32

history = model.fit(X_train, y_train, epochs=15, batch_size=BATCH_SIZE, shuffle=False, validation_split=0.1)
model.evaluate(X_test, y_test)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
y_hat = model.predict(X_test)

y_test_inverse = scaler.inverse_transform(y_test)
y_hat_inverse = scaler.inverse_transform(y_hat)

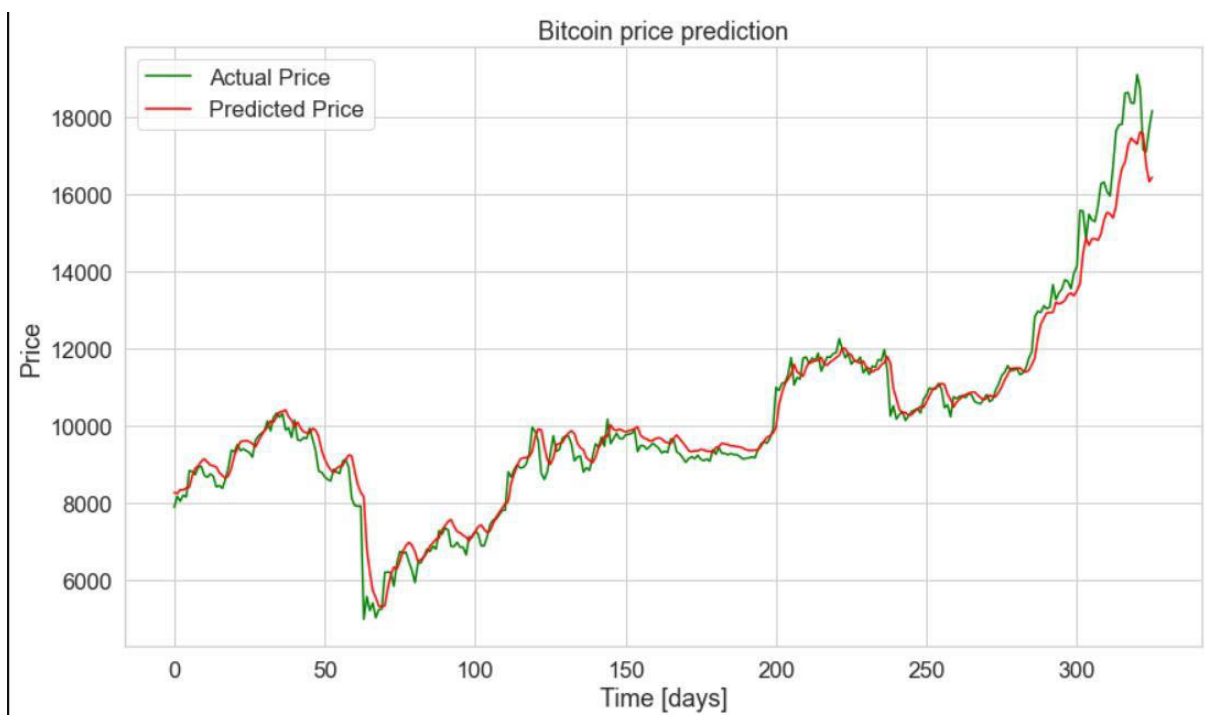
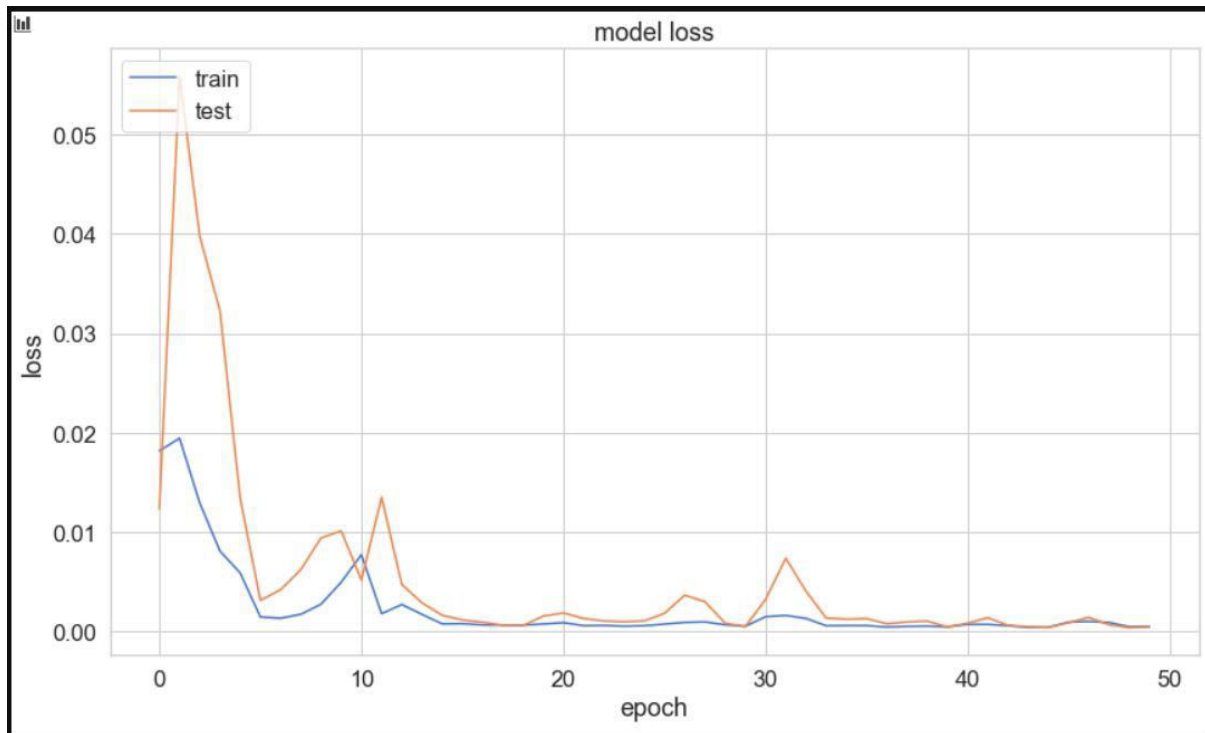
plt.plot(y_test_inverse, label="Actual Price", color='green')
plt.plot(y_hat_inverse, label="Predicted Price", color='red')

plt.title('Bitcoin price prediction')
plt.xlabel('Time [days]')
plt.ylabel('Price')
plt.legend(loc='best')

plt.show()

```

Results:



Conclusion:

Thus, we have learned how to code LSTMs and understood How Recurrent neural networks work

Assignment No. 8

Aim: Develop an autoencoder to encode and decode the image. Analyze the results.

- a) Develop AE for MNIST dataset
- b) Use output of AE as input to CNN

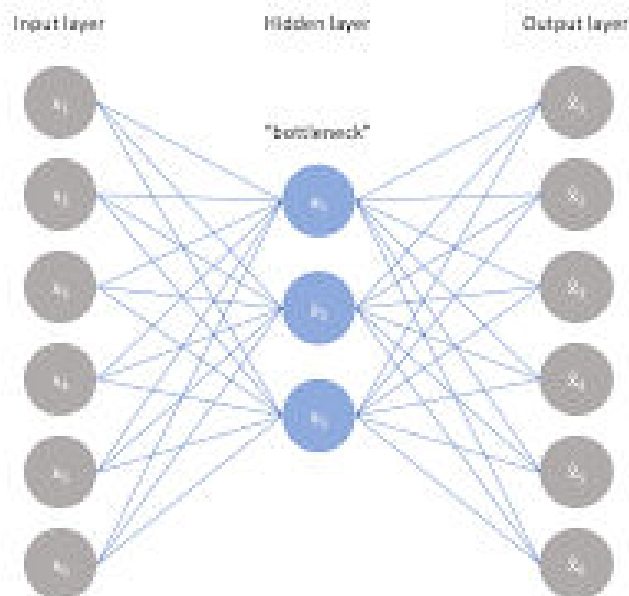
Objectives:

- 1. To learn AE
- 2. To implement AE

Theory:

Autoencoders

An **autoencoder** is a type of artificial neural network used to learn efficient data coding in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. Several variants exist to the basic model, with the aim of forcing the learned representations of the input to assume useful properties. Examples are the regularized autoencoders (*Sparse*, *Denoising* and *Contractive* autoencoders), proven effective in learning representations for subsequent classification tasks, and *Variational* autoencoders, with their recent applications as generative models. Autoencoders are effectively used for solving many applied problems, from face recognition to acquiring the semantic meaning of words.



There are, basically, 7 types of autoencoders:

- Denoising autoencoder
- Sparse Autoencoder
- Deep Autoencoder
- Contractive Autoencoder
- Undercomplete Autoencoder
- Convolutional Autoencoder
- Variational Autoencoder

1) Denoising Autoencoder

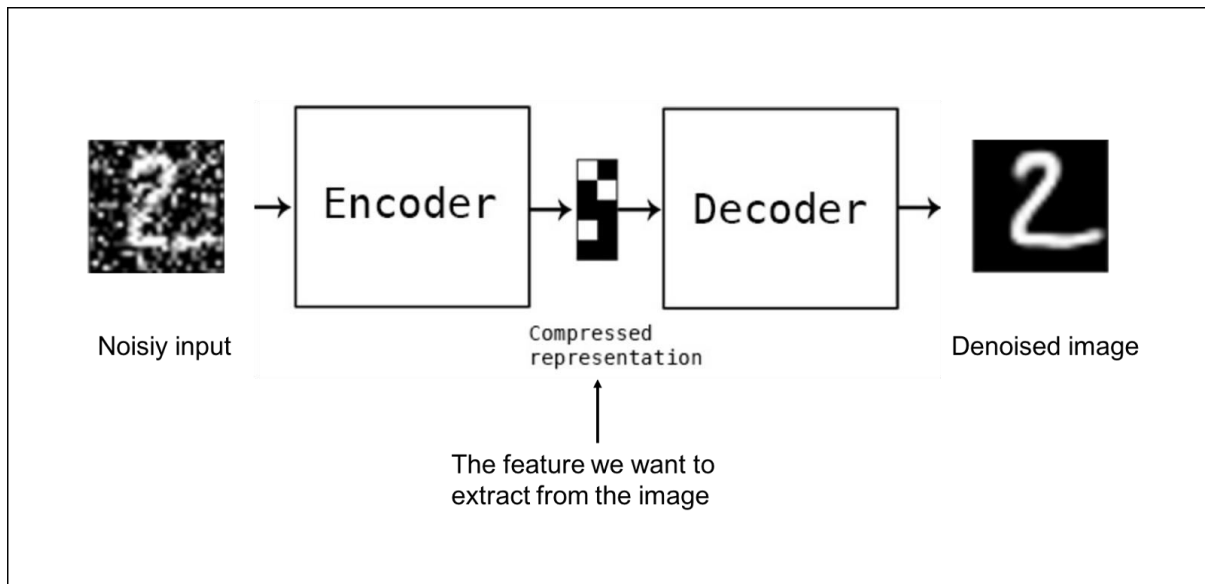
Denoising autoencoders create a corrupted copy of the input by introducing some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data. These autoencoders take a partially corrupted input while training to recover the original undistorted input. The model learns a vector field for mapping the input data towards a lower dimensional manifold which describes the natural data to cancel out the added noise.

Advantages-

- It was introduced to achieve good representation. Such a representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input.
- Corruption of the input can be done randomly by making some of the input as zero. Remaining nodes copy the input to the noised input.
- Minimizes the loss function between the output node and the corrupted input.
- Setting up a single-thread denoising autoencoder is easy.

Drawbacks-

- To train an autoencoder to denoise data, it is necessary to perform preliminary stochastic mapping in order to corrupt the data and use as input.
- This model isn't able to develop a mapping which memorizes the training data because our input and target output are no longer the same.



2) Sparse Autoencoder

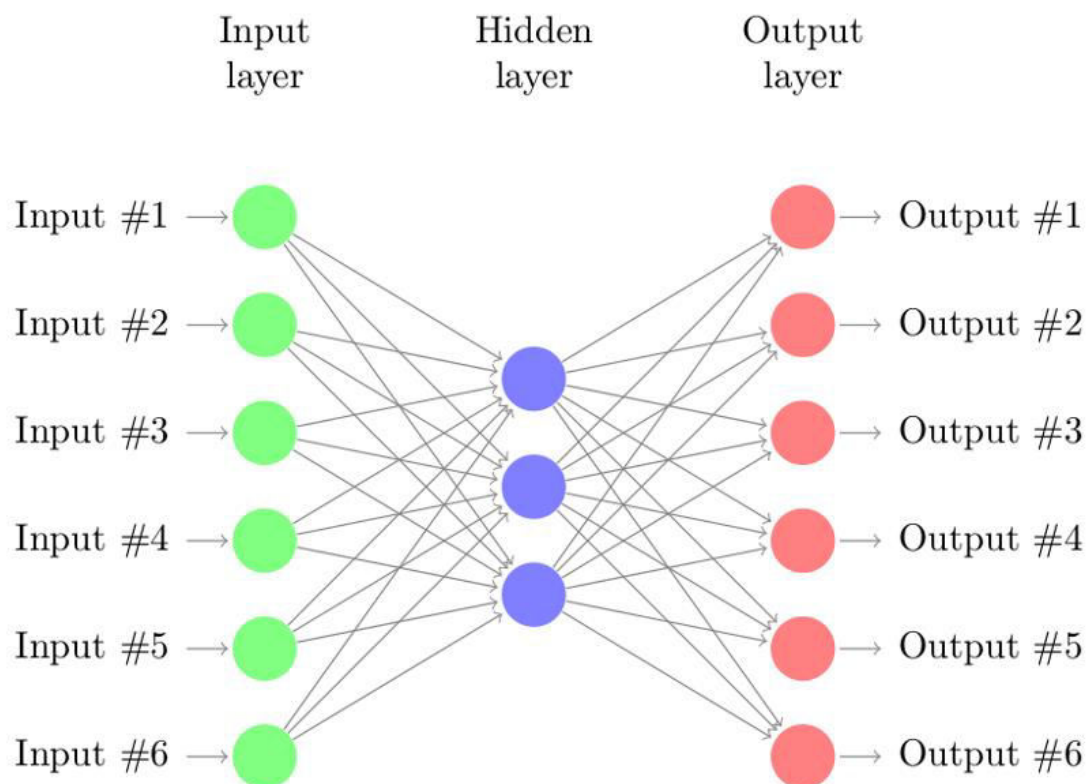
Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data. A generic sparse autoencoder is visualized where the obscurity of a node corresponds with the level of activation. Sparsity constraint is introduced on the hidden layer. This is to prevent output layer copy input data. Sparsity may be obtained by additional terms in the loss function during the training process, either by comparing the probability distribution of the hidden unit activations with some low desired value, or by manually zeroing all but the strongest hidden unit activations. Some of the most powerful AIs in the 2010s involved sparse autoencoders stacked inside of deep neural networks.

Advantages-

- Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer in addition to the reconstruction error. This prevents overfitting.
- They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all of the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.

Drawbacks-

- For it to be working, it's essential that the individual nodes of a trained model which activate are data dependent, and that different inputs will result in activations of different nodes through the network.



3) Deep Autoencoder

Deep Autoencoders consist of two identical deep belief networks, one network for encoding and another for decoding. Typically deep autoencoders have 4 to 5 layers for encoding and the next 4 to 5 layers for decoding. We use unsupervised layer by layer pre-training for this model. The layers are Restricted Boltzmann Machines which are the building blocks of deep-belief networks. Processing the benchmark dataset MNIST, a deep autoencoder would use binary transformations after each RBM. Deep autoencoders are useful in topic modeling, or statistically modeling abstract topics that are distributed across a collection of documents. They are also capable of compressing images into 30 number vectors.

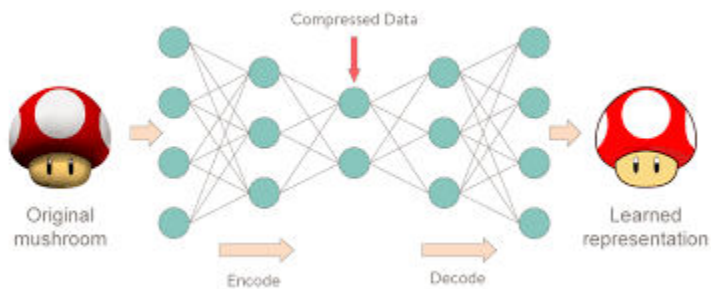
Advantages-

- Deep autoencoders can be used for other types of datasets with real-valued data, on which you would use Gaussian rectified transformations for the RBMs instead.
- Final encoding layer is compact and fast.

Drawbacks-

- Chances of overfitting to occur since there's more parameters than input data.

- Training the data maybe a nuance since at the stage of the decoder's backpropagation, the learning rate should be lowered or made slower depending on whether binary or continuous data is being handled.

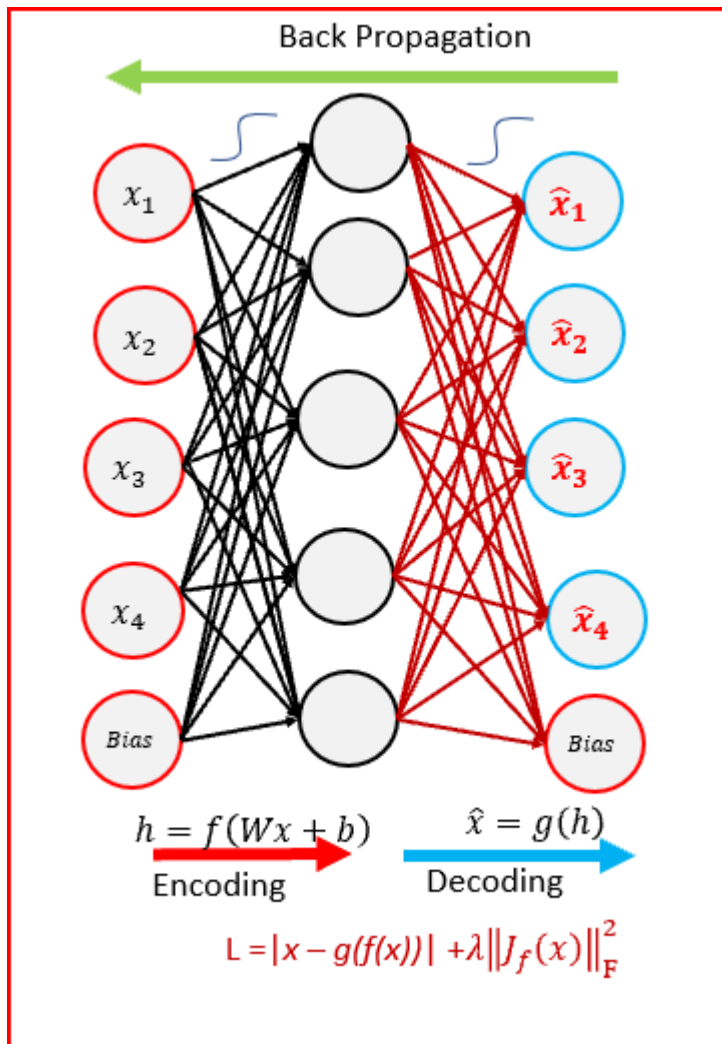


4) Contractive Autoencoder

The objective of a contractive autoencoder is to have a robust learned representation which is less sensitive to small variation in the data. Robustness of the representation for the data is done by applying a penalty term to the loss function. Contractive autoencoder is another regularization technique just like sparse and denoising autoencoders. However, this regularizer corresponds to the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input. Frobenius norm of the Jacobian matrix for the hidden layer is calculated with respect to input and it is basically the sum of square of all elements.

Advantages-

- Contractive autoencoder is a better choice than denoising autoencoder to learn useful feature extraction.
- This model learns an encoding in which similar inputs have similar encodings. Hence, we're forcing the model to learn how to contract a neighborhood of inputs into a smaller neighborhood of outputs.



5) Undercomplete Autoencoder

The objective of undercomplete autoencoder is to capture the most important features present in the data. Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer. This helps to obtain important features from the data. It minimizes the loss function by penalizing the $g(f(x))$ for being different from the input x .

Advantages-

- Undercomplete autoencoders do not need any regularization as they maximize the probability of data rather than copying the input to the output.

Drawbacks-

- Using an overparameterized model due to lack of sufficient training data can create overfitting.

6) Convolutional Autoencoder

Autoencoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders use the convolution operator to exploit

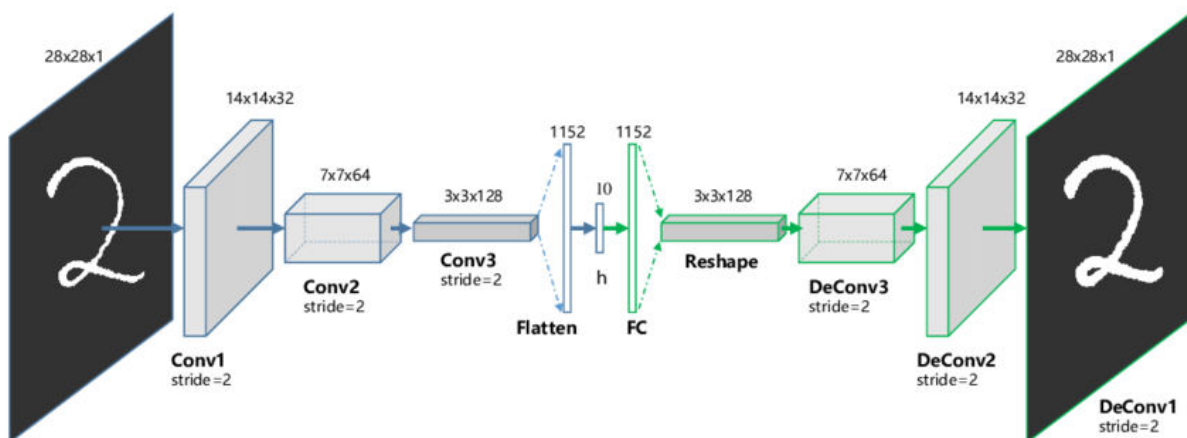
this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them, modify the geometry or the reflectance of the image. They are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.

Advantages-

- Due to their convolutional nature, they scale well to realistic-sized high dimensional images.
- Can remove noise from picture or reconstruct missing parts.

Drawbacks-

- The reconstruction of the input image is often blurry and of lower quality due to compression during which information is lost.



7) Variational Autoencoder

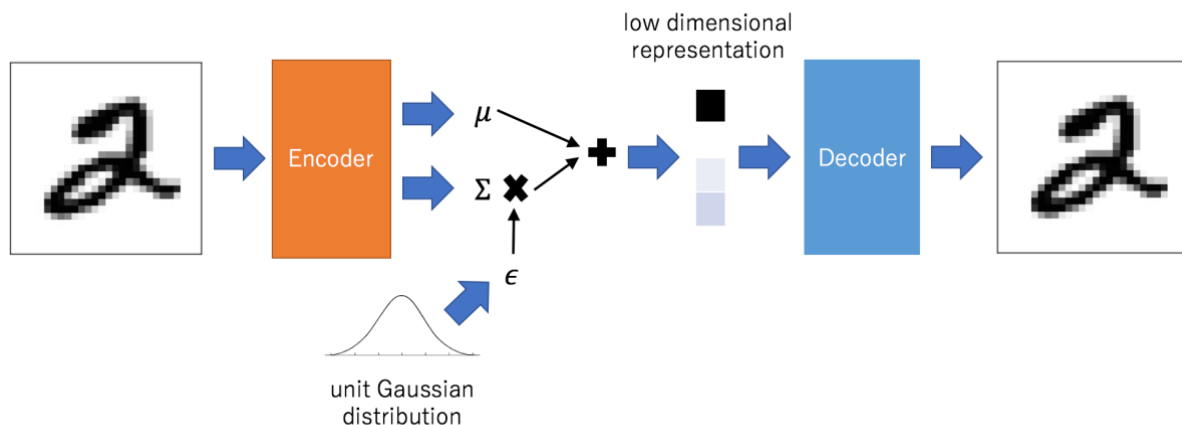
Variational autoencoder models make strong assumptions concerning the distribution of latent variables. They use a variational approach for latent representation learning, which results in an additional loss component and a specific estimator for the training algorithm called the Stochastic Gradient Variational Bayes estimator. It assumes that the data is generated by a directed graphical model and that the encoder is learning an approximation to the posterior distribution where Φ and θ denote the parameters of the encoder (recognition model) and decoder (generative model) respectively. The probability distribution of the latent vector of a variational autoencoder typically matches that of the training data much closer than a standard autoencoder.

Advantages-

- It gives significant control over how we want to model our latent distribution unlike the other models.
- After training you can just sample from the distribution followed by decoding and generating new data.

Drawbacks-

- When training the model, there is a need to calculate the relationship of each parameter in the network with respect to the final output loss using a technique known as backpropagation. Hence, the sampling process requires some extra attention.



Applications

Autoencoders work by compressing the input into a latent space representation and then reconstructing the output from this representation. This kind of network is composed of two parts:

1. Encoder: This is the part of the network that compresses the input into a latent-space representation. It can be represented by an encoding function $h=f(x)$.
2. Decoder: This part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function $r=g(h)$.

If the only purpose of autoencoders was to copy the input to the output, they would be useless. We hope that by training the autoencoder to copy the input to the output, the latent representation will take on useful properties. This can be achieved by creating constraints on the copying task. If the autoencoder is given too much capacity, it can learn to perform the copying task without extracting any useful information about the distribution of the data. This can also occur if the dimension of the latent representation is the same as the input, and in the overcomplete case, where the dimension of the latent representation is greater than the input. In these cases, even a linear encoder and linear decoder can learn to copy the input to the output without learning anything useful about the data distribution. Ideally, one could train any architecture of autoencoder successfully, choosing the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be modeled.

Autoencoders are learned automatically from data examples. It means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input and that it does not require any new engineering, only the appropriate training data. However, autoencoders will do a poor job for image compression. As the autoencoder is trained on a given set of data, it will achieve reasonable compression results on data similar to the training set used but will be poor

general-purpose image compressors. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties.

Code:

```
from keras.datasets import mnist
from keras.layers import Input, Dense
from keras.models import Model
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
(X_train, _), (X_test, _) = mnist.load_data()

X_train = X_train.astype('float32')/255
X_test = X_test.astype('float32')/255

X_train = X_train.reshape(len(X_train), np.prod(X_train.shape[1:]))
X_test = X_test.reshape(len(X_test), np.prod(X_test.shape[1:]))
print(X_train.shape)
print(X_test.shape)
input_img= Input(shape=(784,))
encoded = Dense(units=32, activation='relu')(input_img)
decoded = Dense(units=784, activation='sigmoid')(encoded)
autoencoder=Model(input_img, decoded)
encoder = Model(input_img, encoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=
['accuracy'])
autoencoder.fit(X_train, X_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(X_test, X_test))

encoded_imgs = encoder.predict(X_test)
predicted = autoencoder.predict(X_test)

plt.figure(figsize=(40, 4))
for i in range(10):
    # display original
    ax = plt.subplot(3, 20, i + 1)
    plt.imshow(X_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```

```

# display encoded image
ax = plt.subplot(3, 20, i + 1 + 20)
plt.imshow(encoded_imgs[i].reshape(8,4))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# display reconstruction
ax = plt.subplot(3, 20, 2*20 + i + 1)
plt.imshow(predicted[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

```

plt.show()

Results:



Conclusion:

Thus, we have understood how autoencoders work and how to program them

