DRONACHARYA
Group of Institutions

B-27, Knowledge Park – III, Greater Noida Uttar Pradesh - 201308
Approved by: All India Council for Technical Education (AICTE), New Delhi
Affiliated to: Dr. A. P. J. Abdul Kalam Technical University (AKTU), Lucknow

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEEP LEARNING LAB

SUBJECT CODE: KCS-078

B.Tech., Semester -VII

Session: 2024-25, ODD Semester

Table of Contents

DRONACHARYA GROUP OF INSTITUTIONS GREATER NOIDA

VISION

- Instilling core human values and facilitating competence to address global challenges by providing Quality Technical Education.

MISSION

- M1 - Enhancing technical expertise through innovative research and education, fostering creativity and excellence in problem-solving.
- M2 - Cultivating a culture of ethical innovation and user-focused design, ensuring technological progress enhances the well-being of society.
- M3 - Equipping individuals with the technical skills and ethical values to lead and innovate responsibly in an ever-evolving digital land.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

Promoting technologists by imparting profound knowledge in information technology, all while instilling ethics through specialized technical education.

MISSION

- Delivering comprehensive knowledge in information technology, preparing technologists to excel in a rapidly evolving digital landscape.
- Building a culture of honesty and responsibility in tech, promoting smart and ethical leadership.
- Empowering individuals with specialized technical skills and ethical values to drive positive change and innovation in the tech industry.

Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: 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.

PO3: 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.

PO4: 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.

PO5: 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.

PO6: 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.

PO7: 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.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

PO 9: Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply theseto one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long 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.

Programme Educational Objectives (PEOs)

PEO1: To enable graduates to pursue higher education and research, or have a successful career in industries associated with Computer Science and Engineering, or as entrepreneurs.

PEO2: To ensure that graduates will have the ability and attitude to adapt to emerging technological changes.

PEO3: To prepare students to analyze existing literature in an area of specialization and ethically develop innovative methodologies to solve the problems identified.

Program Specific Outcomes (PSOs)

PSO1: To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO2: To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO3: To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems.

Deep Learning Lab Manual		
S. No.	**Name of the Program**	**Page No.**
1	Design a single unit perceptron for classification of a linearly separable binary dataset without using pre-defined models. Use the Perceptron() fromsklearn.	
2	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. Vary the activation functions used and compare the results.	
3	Build a Deep Feed Forward ANN by implementing the Backpropagation algorithm and test the same using appropriate data sets.	
4	Design and implement an Image classification model to classify a dataset of images using Deep Feed Forward NN Use the MNIST, CIFAR-10 datasets.	
5	Design and implement a CNN model (with 2 layers of convolutions) to classify multi category image datasets.	
6	Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Use the MNIST, Fashion MNIST, CIFAR-10 datasets. Set the No. of Epoch as 5, 10 and 20. Make the necessary changes whenever required.	
7	Design and implement a CNN model (with 2+ layers of convolutions) to classify multi category image datasets. Use the concept of padding and Batch Normalization while designing the CNN model.	
8	Use the concept of Data Augmentation to increase the data size from a singleimage.	
9	Implement the standard VGG 16 CNN architecture model to classify cat and dog image dataset .	
10	Implement RNN for sentiment analysis on movie reviews	

Cos	COURSE OUTCOMES
KCS078.1	Understand the basic concepts and techniques of Deep Learning and the need of Deep Learning techniques in real-world problems.
KCS078.2	Understand CNN algorithms and the way to evaluate performance of the CNN architectures.
KCS078.3	Understand and implement algorithm to solve problems by Greedy algorithm approach
KCS078.4	Apply RNN and LSTM to learn, predict and classify the real-world problems in the paradigms of Deep Learning
KCS078.5	Understand, learn and design GANs for the selected problems.

Mapping of Program Outcomes with Course Outcomes (COs)

CO-PO Matrix												
Course Outcomes	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
BCS-553.1	2		3	3		-	-	-	-	-	-	2
BCS-553.2		3	3	2	2	-	-	-	-	-	-	3
BCS-553.3	2	3	3	3	3	-	-	-	-	-	-	2
BCS-553.4	2		2		2	-	-	-	-	-	-	2
BCS-553.5	2	3		2	2	-	-	-	-	-	-	3

CO-PSO Matrix			
COs	**PSO1**	**PSO2**	**PSO3**
BCS-553.1	**1**	**2**	**1**
BCS-553.2		**3**	**1**
BCS-553.3	**1**	**2**	**1**
BCS-553.4		**1**	**1**

BCS-553.5	1	3

1. **Design a single unit perceptron for classification of a linearly separable binary dataset without using pre-defined models. Use the Perceptron() fromsklearn.**

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron
df=pd.read_csv('/content/gdrive/My Drive/ML_lab/placement.csv') X = df.iloc[:,0:2]
y = df.iloc[:,-1] p = Perceptron() p.fit(X,y) print(p.coef_) print(p.intercept_)
z=p.score(X,y)
print("accuracy score is",z)
from mlxtend.plotting import plot_decision_regions plot_decision_regions(X.values,
y.values, clf=p, legend=2)
```

2. **Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. Vary the activation functions used and compare the results.**

Program:

```
from keras.models import Sequential
from keras.layers import Dense, Activation import numpy as np
import pandas as pd
from sklearn import datasets iris = datasets.load_iris()
X, y = datasets.load_iris( return_X_y = True)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.40) # Define the network model and its arguments.
# Set the number of neurons/nodes for each layer:
model = Sequential() model.add(Dense(2, input_shape=(4,))) model.add(Activation('sigmoid')) model.add(Dense(1)) model.add(Activation('sigmoid'))
#sgd = SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True) #model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
# Compile the model and calculate its accuracy: model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
#model.fit(X_train, y_train, batch_size=32, epochs=3)
# Print a summary of the Keras model: model.summary()
#model.fit(X_train, y_train)
#model.fit(X_train, y_train, batch_size=32, epochs=300) model.fit(X_train, y_train, epochs=5)
score = model.evaluate(X_test, y_test) print(score)
```

3. **Design and implement an Image classification model to classify a dataset of images using Deep Feed ForwardNN. Record the accuracy corresponding to the number of epochs. Use the MNIST datasets**

```
#load required packages import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential from keras import Input
from keras.layers import Dense import pandas as pd
import numpy as np import sklearn
from sklearn.metrics import classification_report import matplotlib
import matplotlib.pyplot as plt

# Load digits data
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Print shapes
print("Shape of X_train: ", X_train.shape) print("Shape of y_train: ", y_train.shape)
print("Shape of X_test: ", X_test.shape) print("Shape of y_test: ", y_test.shape)

# Display images of the first 10 digits in the training set and their true lables fig, axs =
plt.subplots(2, 5, sharey=False, tight_layout=True, figsize=(12,6), facecolor='white')
n=0
for i in range(0,2):
for j in range(0,5): axs[i,j].matshow(X_train[n]) axs[i,j].set(title=y_train[n]) n=n+1
plt.show()

# Reshape and normalize (divide by 255) input data
X_train = X_train.reshape(60000, 784).astype("float32") / 255 X_test =
X_test.reshape(10000, 784).astype("float32") / 255

# Print shapes
print("New shape of X_train: ", X_train.shape) print("New shape of X_test: ",
X_test.shape)

#Design the Deep FF Neural Network architecture model = Sequential(name="DFF-
Model") # Model
model.add(Input(shape=(784,), name='Input-Layer')) # Input Layer - need to specify
the shape of inputs
model.add(Dense(128, activation='relu', name='Hidden-Layer-1',
kernel_initializer='HeNormal'))
model.add(Dense(64, activation='relu', name='Hidden-Layer-2',
kernel_initializer='HeNormal'))
model.add(Dense(32, activation='relu', name='Hidden-Layer-3',
kernel_initializer='HeNormal'))
model.add(Dense(10, activation='softmax', name='Output-Layer'))

#Compile keras model
model.compile(optimizer='adam', loss='SparseCategoricalCrossentropy',
```

```
metrics=['Accuracy'], loss_weights=None, weighted_metrics=None, run_eagerly=None, steps_per_execution=None)

#Fit keras model on the dataset
model.fit(X_train, y_train, batch_size=10, epochs=5, verbose='auto', callbacks=None, validation_split=0.2, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, # Integer, default=0, Epoch at which to start training (useful for resuming a previous training run).
steps_per_epoch=None, validation_steps=None, validation_batch_size=None, validation_freq=5, max_queue_size=10, workers=1, use_multiprocessing=False,)

# apply the trained model to make predictions # Predict class labels on training data
pred_labels_tr = np.array(tf.math.argmax(model.predict(X_train),axis=1)) # Predict class labels on a test data
pred_labels_te = np.array(tf.math.argmax(model.predict(X_test),axis=1))



#Model Performance Summary print("")
print('  Model Summary model.summary()
print("")



        ')


# Printing the parameters:Deep Feed Forward Neural Network contains more than 100K

#print('Weights and Biases #for layer in model_d1.layers:

        ')

#print("Layer: ", layer.name) # print layer name
#print(" --Kernels (Weights): ", layer.get_weights()[0]) # kernels (weights) #print(" --Biases: ", layer.get_weights()[1]) # biases

print("")
print('---------- Evaluation on Training Data  ')
print(classification_report(y_train, pred_labels_tr)) print("")

print('---------- Evaluation on Test Data        ')
print(classification_report(y_test, pred_labels_te)) print("")
```

4. **Design and implement a CNN model (with 2 layers of convolutions) to classify multi category image datasets. Use the MNIST, CIFAR-10 datasets.**

```
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
import matplotlib.pyplot as plt

# Load the dataset (Change between MNIST and CIFAR-10 as needed)
dataset_name = "MNIST"  # Change to "CIFAR-10" for CIFAR dataset

if dataset_name == "MNIST":
    (X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()
    X_train = X_train.reshape(-1, 28, 28, 1) / 255.0
    X_test = X_test.reshape(-1, 28, 28, 1) / 255.0
    num_classes = 10
    input_shape = (28, 28, 1)
elif dataset_name == "CIFAR-10":
    (X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
    X_train = X_train / 255.0
    X_test = X_test / 255.0
    num_classes = 10
    input_shape = (32, 32, 3)

# One-hot encode labels
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

# Define the CNN model
def create_cnn_model(input_shape, num_classes):
    model = models.Sequential([
        # 1st Convolutional Layer
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        # 2nd Convolutional Layer
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # Flatten and Dense layers
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Create the CNN model
```

```
model = create_cnn_model(input_shape, num_classes)

# Train the model and record accuracy for different epochs
epochs = 10  # Number of epochs
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs,
batch_size=32)

# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

5. **Design and implement a CNN model (with 4+ layers of convolutions) to classify multi category image datasets. Record the accuracy corresponding to the number of epochs. Use the Fashion MNIST datasets**

```
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D from
keras.models import Sequential
from keras.utils import to_categorical import numpy as np
import matplotlib.pyplot as plt
(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data() train_X =
train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32') test_X = test_X.astype('float32') train_X = train_X
/ 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y) test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=(28, 28, 1))) model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(128, (3,3))) model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1))) model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(28, (3,3))) model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten()) model.add(Dense(64))

model.add(Dense(10)) model.add(Activation('softmax'))
```

6. **Design and implement a CNN model (with 2+ layers of convolutions) to classify multi category image datasets. Use the concept of padding and Batch Normalization while designing the CNN model.**

```
# Batch-Normalization and padding

import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D,
BatchNormalization
from keras.models import Sequential from keras.utils import to_categorical import
numpy as np
import matplotlib.pyplot as plt
(train_X,train_Y), (test_X,test_Y) = fashion_mnist.load_data() train_X =
train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)

train_X = train_X.astype('float32') test_X = test_X.astype('float32') train_X = train_X
/ 255
test_X = test_X / 255

train_Y_one_hot = to_categorical(train_Y) test_Y_one_hot = to_categorical(test_Y)

model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=(28, 28, 1),padding='same'))
model.add(Activation('relu'))
BatchNormalization() model.add(MaxPooling2D(pool_size=(2,2) ,padding='same'))

model.add(Conv2D(128, (3,3),padding='same')) model.add(Activation('relu'))
#BatchNormalization()
model.add(MaxPooling2D(pool_size=(2,2) ,padding='same'))

model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1,padding='same'))
model.add(Activation('relu'))
#BatchNormalization() model.add(MaxPooling2D(pool_size=(2,2),padding='same'))

model.add(Conv2D(28, (3,3))) model.add(Activation('relu'))
```

7. **Use the concept of Data Augmentation to increase the data size from a single image**.

```
#data augmentation on a single image

from numpy import expand_dims
from tensorflow.keras.preprocessing import image #from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator from matplotlib import pyplot
# load the image
img = image.load_img('/content/gdrive/My Drive/data/train/cat.png') # convert to numpy array
data = image.img_to_array(img) # expand dimension to one sample samples = expand_dims(data, 0)
# create image data augmentation generator
datagen = ImageDataGenerator(width_shift_range=[-100,100]) # prepare iterator
it = datagen.flow(samples, batch_size=1) # generate samples and plot
for i in range(9):
# define subplot pyplot.subplot(330 + 1 + i) # generate batch of images batch = it.next()
# convert to unsigned integers for viewing image = batch[0].astype('uint8')
# plot raw pixel data pyplot.imshow(image)
# show the figure
pyplot.show()
```

8. **Design and implement a CNN model to classify CIFAR10 image dataset. Use the concept of Data Augmentationwhile designing the CNN model.**

```
# data augmentation with flow function from future import print_function
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator from
tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten from
tensorflow.keras.layers import Conv2D, MaxPooling2D

import matplotlib.pyplot as plt
%matplotlib inline

# The data, shuffled and split between train and test sets: (x_train, y_train), (x_test,
y_test) = cifar10.load_data() print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples') print(x_test.shape[0], 'test samples')

num_classes = 10

y_train = tf.keras.utils.to_categorical(y_train, num_classes) y_test =
tf.keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32') x_test = x_test.astype('float32') x_train /= 255
x_test /= 255

# Let's build a CNN using Keras' Sequential capabilities model_1 = Sequential()

## 5x5 convolution with 2x2 stride and 32 filters model_1.add(Conv2D(32, (5, 5),
strides = (2,2), padding='same',
input_shape=x_train.shape[1:])) model_1.add(Activation('relu'))

## Another 5x5 convolution with 2x2 stride and 32 filters model_1.add(Conv2D(32,
(5, 5), strides = (2,2))) model_1.add(Activation('relu'))

## 2x2 max pooling reduces to 3 x 3 x 32 model_1.add(MaxPooling2D(pool_size=(2,
2))) model_1.add(Dropout(0.25))

## Flatten turns 3x3x32 into 288x1 model_1.add(Flatten()) model_1.add(Dense(512))
model_1.add(Activation('relu')) model_1.add(Dropout(0.5))
model_1.add(Dense(num_classes)) model_1.add(Activation('softmax'))

model_1.summary() batch_size = 32
# initiate RMSprop optimizer
opt = tf.keras.optimizers.legacy.RMSprop(lr=0.0005, decay=1e-6)

# Let's train the model using RMSprop
model_1.compile(loss='categorical_crossentropy',
optimizer=opt, metrics=['accuracy'])
```

```
datagen = ImageDataGenerator(
featurewise_center=False, # set input mean to 0 over the dataset
samplewise_center=False, # set each sample mean to 0
featurewise_std_normalization=False, # divide inputs by std of the dataset
samplewise_std_normalization=False, # divide each input by its std
zca_whitening=False, # apply ZCA whitening
rotation_range=0, # randomly rotate images in the range (degrees, 0 to 180)
width_shift_range=0.1, # randomly shift images horizontally (fraction of total
width)
height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
horizontal_flip=True, # randomly flip images vertical_flip=False) # randomly flip
images

datagen.fit(x_train)     # This computes any statistics that may be needed (e.g. for
centering) from the training set.


# Fit the model on the batches generated by datagen.flow().
model_1.fit(datagen.flow(x_train, y_train,
batch_size=batch_size), steps_per_epoch=x_train.shape[0] // batch_size, epochs=5,
validation_data=(x_test, y_test)) test_loss, test_acc = model_1.evaluate(x_test, y_test)
```

9. Implement the standard VGG 16 CNN architecture model to classify cat and dog image dataset .

```
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten from
keras.preprocessing.image import ImageDataGenerator import numpy as np
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="/content/gdrive/My
Drive/training_set",target_size=(224,224))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="/content/gdrive/My Drive/test_set",
target_size=(224,224))

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="s
ame"
,activation="relu")) model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",
activation="relu")) model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",
activation="relu")) model.add(Conv2D(filters=128, kernel_size=(3,3),
padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",
activation="relu")) model.add(Conv2D(filters=256, kernel_size=(3,3),
padding="same", activation="relu")) model.add(Conv2D(filters=256,
kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu")) model.add(Conv2D(filters=512, kernel_size=(3,3),
padding="same", activation="relu")) model.add(Conv2D(filters=512,
kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",
activation="relu")) model.add(Conv2D(filters=512, kernel_size=(3,3),
padding="same", activation="relu")) model.add(Conv2D(filters=512,
kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten()) model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu")) model.add(Dense(units=2,
activation="softmax"))

from keras.optimizers import Adam opt = Adam(lr=0.001)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy,
metrics=['accuracy'])

model.summary()
```

```
from keras.callbacks import ModelCheckpoint, EarlyStopping checkpoint =
ModelCheckpoint("vgg16_1.h5", monitor='val_acc', verbose=1,save_best_only=True,
save_weights_only=False, mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=20, verbose=1,
mode='auto')
hist = model.fit_generator(steps_per_epoch=100,generator=traindata,
validation_data= testdata, validation_steps=10,epochs=5,callbacks=[checkpoint,early])

import matplotlib.pyplot as plt plt.plot(hist.history["accuracy"])
plt.plot(hist.history['val_accuracy']) plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss']) plt.title("model accuracy") plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()
```

10. **Implement RNN for sentiment analysis on movie reviews.**

RNN sentiment analysis on movie reviews

```
from keras.datasets import imdb
from keras.preprocessing.text import Tokenizer from keras.utils import
pad_sequences
from keras import Sequential from keras.layers import
Dense,SimpleRNN,Embedding,Flatten(X_train,y_train),(X_test
,y_test) = imdb.load_data() X_train =
pad_sequences(X_train,padding='post',maxlen=50) X_test =
pad_sequences(X_test,padding='post',maxlen=50)
X_train.shape
model = Sequential() #model.add(Embedding(10000, 2))
model.add(SimpleRNN(32,input_shape=(50,1), return_sequences=False))
model.add(Dense(1, activation='sigmoid'))

model.summary()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(X_train, y_train,epochs=5,validation_data=(X_test,y_test)) test_loss,
test_acc = model.evaluate(X_test, y_test)
print('Test loss', test_loss) print('Test accuracy', test_acc)
```