FEED FORWARD –MNIST

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
from tensorflow.keras.layers import BatchNormalization, Dense, Dropout, MaxPooling2D, Conv2D
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.utils import to_categorical


train = pd.read_csv("mnist_train.csv")
train

test = pd.read_csv("/content/mnist_test.csv")
test

train.isnull().sum().sum()

x_train = train.drop(['label'], axis=1).values
x_train
```

```
x_train = x_train.astype('float32')/255
y_train = train['label'].values


# c. Define the network architecture using Keras
model = Sequential()
model.add(Dense(128, input_shape = (784, ) ,activation = 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
model.compile(optimizer= 'adam', loss =
'sparse_categorical_crossentropy', metrics = ['accuracy'])
model.summary()


history = model.fit(x_train, y_train, epochs=11, batch_size=32,
validation_split=0.2)
r = model.fit(x_train, y_train, validation_split= 0.2, batch_size = 128,
epochs = 11)

x_test = test.drop(['label'], axis = 1).values
```

```
y_test = test['label'].values

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

test_loss, test_accuracy = model.evaluate(x_test, y_test)
plt.plot(r.history['accuracy'], label = 'accuracy', color = 'green')
plt.plot(r.history['val_accuracy'], label = 'val_accuracy', color = 'red')
plt.legend()


plt.plot(r.history['loss'], label = 'loss', color = 'red')
plt.plot(r.history['val_loss'], label = 'val_loss', color = 'blue')
plt.legend()
```

FEED FORWARD - CIFAR

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.losses import SparseCategoricalCrossentropy

train = pd.read_csv('train_data.csv')
test = pd.read_csv('test_data.csv')

train
test
x_train = train.drop(['label'],axis = 1).values
y_train = train['label'].values

x_test = test.drop(['label'],axis = 1).values
y_test = test['label'].values
x_test

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

x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
x_train = x_train.reshape(x_train.shape[0],-1)
x_test = x_test.reshape(x_test.shape[0],-1)

model = Sequential()
model.add(Dense(units = 128,input_shape = (3072,),activation = 'relu'))
model.add(Dense(units = 64,activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 10,activation = 'softmax'))
model.compile(optimizer = 'sgd', metrics = ['accuracy'], loss =
SparseCategoricalCrossentropy())
model.summary()

H = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs =
25)

train_loss, train_accuracy = model.evaluate(x_test,y_test)

import matplotlib.pyplot as plt
plt.plot(H.history['accuracy'],color = 'blue')
plt.plot(H.history['loss'],color = 'red')

plt.plot(H.history['loss'],color = 'blue')
plt.plot(H.history['val_loss'],color = 'red')
```

IMAGE CLASSIFICATION – CIFAR

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

from tensorflow.keras.losses import SparseCategoricalCrossentropy

train = pd.read_csv(”train.csv")
train

x_train = train.drop(['label'], axis = 1).values
y_train = train['label'].values
x_train.shape

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train = x_train / 255.0
x_train = x_train / 255.0
x_train = x_train.reshape((50000,32,32,3))
x_test = x_test.reshape((10000,32,32,3))

test = pd.read_csv("test.csv")
```

test

```
model = Sequential()
model.add(Conv2D(32,(3,3),input_shape=(32,32,3),activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
model.add(Dense(units = 128,activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 10,activation = 'softmax'))
model.compile(optimizer='adam',metrics = ['accuracy'], loss =
SparseCategoricalCrossentropy())
model.summary()

H = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs =
11)

test_loss, test_acc = model.evaluate(x_test,y_test)

plt.plot(H.history['accuracy'],color = 'blue')
plt.plot(H.history['loss'],color = 'red')
plt.grid()
plt.legend()

plt.plot(H.history['loss'],color = 'blue')

plt.plot(H.history['val_loss'],color = 'red')

plt.grid()

plt.legend()
```

IMAGE CLASSIFICATION –MINST

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras.layers import Dense, MaxPooling2D, Flatten, Dropout, Conv2D
from tensorflow.keras.losses import SparseCategoricalCrossentropy

train = pd.read_csv("mnist_train.csv")
test = pd.read_csv('mnist_test.csv')

train
test

x_train = train.drop(['label'],axis = 1).values
y_train = train['label'].values
x_train.shape
```

```
x_test = test.drop(['label'], axis = 1).values
y_test = test['label']
x_test.shape

x_train = x_train.astype('float32')
x_train = x_train/255
x_train[10]

x_test = x_test.astype('float32')
x_test = x_test/255
x_test[10]

x_train = x_train.reshape((60000,28,28,1))
x_test = x_test.reshape((10000,28,28,1))

model = Sequential()
model.add(Conv2D(32,(3,3),input_shape=(28,28,1),activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Flatten())
model.add(Dense(units = 64,activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 10,activation = 'softmax'))
```

```
model.compile(optimizer = 'adam', metrics = ['accuracy'], loss = SparseCategoricalCrossentropy())

model.summary()


H = model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs = 11)


prediction = model.predict(x_test)


test_loss, test_accuracy = model.evaluate(x_test,y_test)

plt.plot(H.history['accuracy'])

plt.plot(H.history['loss'])

plt.legend()


plt.plot(H.history['loss'],color = 'red')

plt.plot(H.history['val_loss'], color = 'blue')

plt.legend()


image_index = 4

plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')

pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))

print(pred.argmax())
```

AUTOENCODER –EGC DATASET

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import BatchNormalization,
MaxPooling2D, Conv2D, Dense, Dropout, Flatten
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import SGD
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report

df = pd.read_csv("ecg_autoencoder_dataset.csv", header = None)
df

X = df.drop([140], axis = 1)
y = df[140]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
scaler = StandardScaler()
X_train =  scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
encoder = Sequential([Dense(64, activation = 'relu', input_shape = (X_train.shape[1],))])
decoder = Sequential([Dense(X_train.shape[1], activation = 'sigmoid')])
```

In [ ]:

```
autoencoder = Sequential([encoder, decoder])
```

In [ ]:

```
autoencoder.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics = ['accuracy'])

r = autoencoder.fit(X_train, X_train, epochs = 100, batch_size = 64, validation_data= (X_test, X_test))


r.history.keys()

plt.plot(r.history['loss'], label = 'loss', color = 'blue')
plt.plot(r.history['val_loss'], label = 'val_loss', color = 'green')
plt.legend
plt.xlabel('epoch')
plt.ylabel('loss')


plt.plot(r.history['accuracy'], label = 'acc', color = 'red')
plt.plot(r.history['val_accuracy'], label = 'val_acc', color = 'green')
plt.legend


loss = autoencoder.evaluate(X_test, X_test)
print(f'Test Loss: {loss}')


decoded_data = autoencoder.predict(X_test)
```

```
mse = np.mean(np.power(X_test - decoded_data, 2), axis = 1)
threshold = np.percentile(mse, 95)

outliers = mse> threshold
print("Confusion Matrix:\n", confusion_matrix(y_test, outliers))
print("Classification report:\n", classification_report(y_test, outliers))


num_outliers = np.sum(outliers)
num_anomalies = np.sum(y_test[outliers] == 1)
print(f'Number of outliers: {num_outliers}')
print(f'Number of anomalies: {num_anomalies}')
```

CBOW

```
file_path = "CBOW.txt"

with open(file_path, 'r') as file:
   file_contents = file.read()

file_contents

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

sentences = file_contents.split('.')
tokenizer = Tokenizer()
```

```
tokenizer.fit_on_texts(sentences)

total_words = len(tokenizer.word_index) + 1
window_size = 3
tokenized_sentences = tokenizer.texts_to_sequences(sentences)

data = pad_sequences([[sentence[j] for j in range(i - window_size, i +
window_size + 1) if j != i and 0 <= j < len(sentence)]
            for sentence in tokenized_sentences
            for i, _ in enumerate(sentence)])

labels = np.array([target_word for sentence in tokenized_sentences for
target_word in sentence])


model = models.Sequential([
  layers.Embedding(input_dim=total_words, output_dim=50,
input_length=window_size * 2),
  layers.GlobalAveragePooling1D(),
  layers.Dense(total_words, activation='softmax')
])


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

# Train the model
model.fit(data, labels, epochs=200)


word_embeddings = model.layers[0].get_weights()[0]

from sklearn.metrics.pairwise import cosine_similarity
```

```
target_word = 'influenza'
target_embedding =
word_embeddings[tokenizer.word_index[target_word]]

similarities = cosine_similarity(target_embedding.reshape(1, -1),
word_embeddings)[0]
most_similar_indices = similarities.argsort()[-5:][::-1]

most_similar_words = [word for word, idx in
tokenizer.word_index.items() if idx in most_similar_indices]

print(f"Most similar words to '{target_word}': {most_similar_words}")
```

OBJECT DETECTION

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

from tensorflow.keras.preprocessing.image import
ImageDataGenerator

dataset_dir = "C:\\Users\\kalya\\Downloads\\caltech-101-
img\\caltech-101-img" #Specifies the directory path where the dataset
is located
dataset_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
)
#normalises the image
```

```
# # here batch_size is the number of images in each batch
batch_size = 2000
dataset_generator = dataset_datagen.flow_from_directory(
    dataset_dir,
    target_size=(64, 64), #resizes the image into 64 by 64 pixel
    batch_size=batch_size, #Sets the batch size for training.
    class_mode='categorical' # labels are one-hot encoded
)

x_train, y_train =  dataset_generator[0]
x_test, y_test = dataset_generator[1]

print(len(x_train))
print(len(x_test))

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications import VGG16

weights_path =
"vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
base_model = VGG16(weights=weights_path, include_top=False,
input_shape=(64, 64, 3))

for layer in base_model.layers:
   layer.trainable = False


x = Flatten()(base_model.output)
# Explanation: This line adds a Flatten layer to the output of the
base_model. The Flatten layer is used to transform the 3D tensor
```

output from the convolutional base (which is usually the output of the last convolutional layer) into a 1D tensor. This flattening step is necessary when transitioning from convolutional layers to densely connected layers.
Example: Suppose the output shape of base_model is (7, 7, 512). This means you have a 3D tensor with dimensions 7x7x512. Applying the Flatten layer converts this 3D tensor into a 1D tensor by unraveling the values along the dimensions. In this case, the resulting 1D tensor would have a size of 7 * 7 * 512 = 25088.

```
x = Dense(64, activation='relu')(x)
predictions = Dense(102, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer="adam", loss='categorical_crossentropy',
metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=64,
epochs=10,validation_data=(x_test, y_test))


base_model = VGG16(weights=weights_path, include_top=False,
input_shape=(64, 64, 3))
# freeze all layers first
for layer in base_model.layers:
  layer.trainable = False
# unfreeze last 4 layers of base model
for layer in base_model.layers[len(base_model.layers) - 2:]:
  layer.trainable = True
# fine-tuning hyper parameters
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
```

```
x = tf.keras.layers.Dropout(0.3)(x)
predictions = Dense(102, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
# training fine tuned model
model.fit(x_train, y_train, batch_size=64, epochs=20,
validation_data=(x_test, y_test))



import matplotlib.pyplot as plt
predicted_value = model.predict(x_test)


n = 887

plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
```