Addition:

```
Import tensorflow as tf
vector1 = tf.constant([1,2,3])
vector2 = tf.constant([4,5,6])
 print(vector1)
 print(vector2)
result = tf.add(vector1, vector2)
print("Result of vector addition:",result)
```

regression model:

```
import numpy as np
 from keras.models import Sequential
 from keras.layers import Dense
 np.random.seed(0)
X = np.random.rand(100, 1)
 y = 2 * X + 1 + np.random.randn(100, 1) * 0.1
model = Sequential()
model.add(Dense(10, input_dim=1, activation='relu'))
 model.add(Dense(1))
 model.compile(loss='mean_squared_error', optimizer='adam')
 model.fit(X, y, epochs=1000, batch_size=10, verbose=0)
mse = model.evaluate(X, y, verbose=0)
print('Mean Squared Error:', mse)
```

feedforward:

```
from tensorflow import keras
input_size = 100 # Number of input features
hidden_units = 32 # Number of units in the hidden layer
```

```
 output_units = 1
model = keras.Sequential([.layers.Input(shape=(input_size,)),
 keras.layers.Dense(units=hidden_units, activation='sigmoid'),
keras.layers.Dense(units=output_units, activation='sigmoid') ])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']) summary
model.summary()
```

perceptron:

```
import tensorflow as tf
 from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])
model = Sequential([Dense(1, input_shape=(2,), activation='sigmoid', use_bias=True)])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
 model.fit(X, y, epochs=1000, verbose=0)
 loss, accuracy = model.evaluate(X, y)
print("Loss:", loss)
print("Accuracy:", accuracy)
 predictions = model.predict(X)
 print("Predictions:", predictions.flatten())
```

CNN:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
 (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))
```

```
 for i in range(25):
 plt.subplot(5,5,i+1)
plt.xticks([])
 plt.yticks([])
plt.grid(False)
 plt.imshow(train_images[i])
plt.xlabel(class_names[train_labels[i][0]])
plt.show()
 model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
 model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
 model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentrop
(from_logits=True), metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
 plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
 print(test_acc)
```

tuning network:

```
import numpy as np
from sklearn.datasets import make_classification
 X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_classes=2, random_state=42)
from scipy.stats import randint
from sklearn.tree import DecisionTreeClassifier
 from sklearn.model_selection import RandomizedSearchCV
param_dist = {"max_depth": [3, None], "max_features": randint(1, 9), "min_samples_leaf": randint(1, 9), "criterion": ["gini", "entropy"]}
 tree = DecisionTreeClassifier()
tree_cv = RandomizedSearchCV(tree, param_dist, cv=5)
tree_cv.fit(X, y)
 print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
 print("Best score is {}".format(tree_cv.best_score_))
```

7. . Implement a Transfer Learning concept in Image Classification.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.applications import VGG16
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Load the VGG16 model pre-trained on ImageNet, excluding the top fully connected layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Define the model
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
```

```
datagen.fit(x_train)

# Train the model
model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=20, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
```

8.Using a pre trained model on Keras for Transfer Learning

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Load the MobileNetV2 model pre-trained on ImageNet, excluding the top fully connected layers
```

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Define the model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)
datagen.fit(x_train)

# Train the model
model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=20, validation_data=(x_test, y_test))
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
print(f"Test accuracy: {test_acc:.4f}")
```

sentimental analysis:

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Embedding, SimpleRNN

max_features = 10000

maxlen = 500

batch_size = 32

print('Loading data...')

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

print(len(x_train), 'train sequences')

print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)')

x_train = pad_sequences(x_train, maxlen=maxlen)

x_test = pad_sequences(x_test, maxlen=maxlen)

print('x_train shape:', x_train.shape)

print('x_test shape:', x_test.shape)

model = Sequential()

model.add(Embedding(max_features, 32))

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
print(model.summary())

print('Training...')

history = model.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
print('Evaluating...')

loss, accuracy = model.evaluate(x_test, y_test)

print('Test Loss:', loss) print('Test Accuracy:', accuracy)
```

LSTM:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, LSTM, RepeatVector
from tensorflow.keras.models import Model
seq_length = 10
num_features = 5
num_samples = 1000
X_train = np.random.randn(num_samples, seq_length, num_features)
input_seq = Input(shape=(seq_length, num_features))
encoder = LSTM(32, activation='relu')(input_seq)
encoded = RepeatVector(seq_length)(encoder)
decoder = LSTM(32, activation='relu', return_sequences=True)(encoded)
output_seq= tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(num_features))(decoder )
autoencoder = Model(input_seq, output_seq)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X_train, X_train, epochs=5, batch_size=32, validation_split=0.2)
X_test = np.random.randn(5, seq_length, num_features)
reconstructed_seqs = autoencoder.predict(X_test)
print("Original Sequence:")
print(X_test[0])
 print("\nReconstructed Sequence:")
print(reconstructed_seqs[0])
```

GAN:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
(train_images, _), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5

def make_generator_model():

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(7*7*256, use_bias=False,input_shape=(100,)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Reshape((7, 7, 256)))

assert model.output_shape == (None, 7, 7, 256)

model.add(tf.keras.layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False))

assert model.output_shape == (None, 7, 7, 128)

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False))

assert model.output_shape == (None, 14, 14, 64)

model.add(tf.keras.layers.BatchNormalization())

 model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False,
activation='tanh'))

assert model.output_shape == (None, 28, 28, 1)

return model

def generate_and_show_image(generator):

noise = tf.random.normal([1, 100])

generated_image = generator.predict(noise)[0, :, :, 0]

plt.imshow(generated_image, cmap='gray')

plt.axis('off') plt.show()

image generator = make_generator_model()

generate_and_show_image(generator)
```

TRAIN A DEEP LEARNING MODEL TO CLASSIFY A GIVEN IMAGE USING PRE – TRAINED MODEL

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Input
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
x = base_model.output x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
layer.trainable = False
model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Loss:", loss) print("Test Accuracy:", accuracy)
```