In [ ]:

```
from keras.datasets import cifar10
from keras.layers import Conv2D, Input, Dense, Dropout, MaxPool2D, UpSampling2D
from keras.models import Model,Sequential
```

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading the dataset

In [ ]:

```
(cifar_train, _), (cifar_test, _) = cifar10.load_data()

size = 32
channel = 3
```

Scaling

In [ ]:

```
cifar_train = cifar_train / 255
cifar_test = cifar_test / 255
```

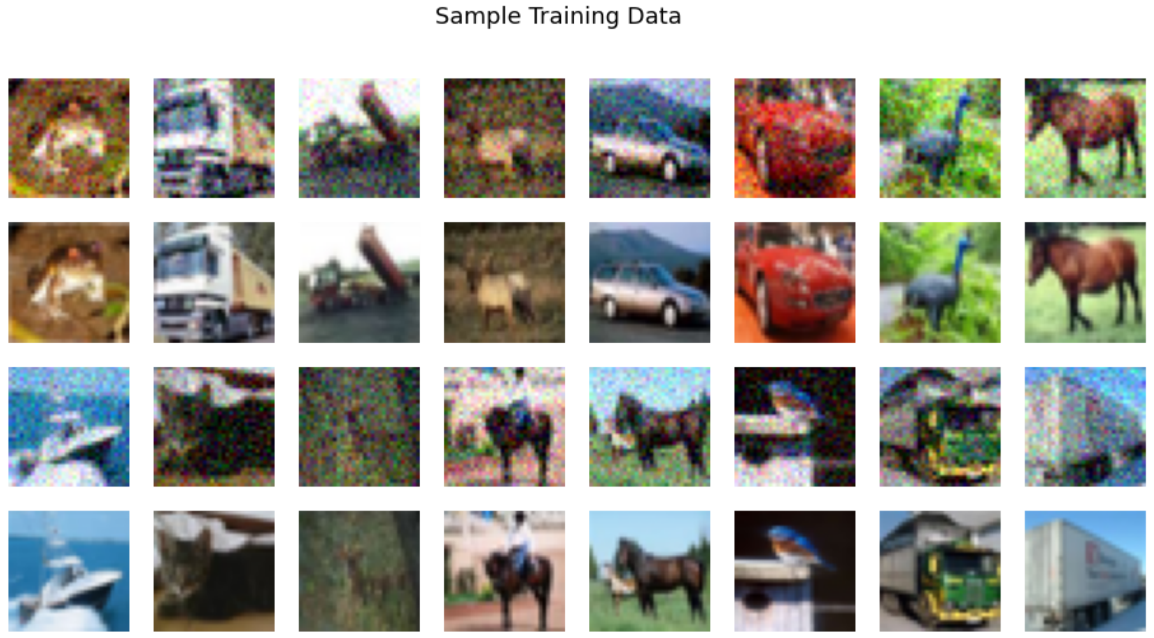
Adding noise

In [ ]:

```
noise = 0.3
cifar_train_noise = cifar_train + noise * np.random.normal(0, 0.3, size=cifar_train.sha
pe)
cifar_test_noise = cifar_test + noise * np.random.normal(0, 0.3, size=cifar_test.shape)

cifar_train_noise = np.clip(cifar_train_noise, 0, 1)
cifar_test_noise = np.clip(cifar_test_noise, 0, 1)
```

In [ ]:

```
rows = 2
cols = 8

f = plt.figure(figsize=(2*cols,2*rows*2))

for i in range(rows):
    for j in range(cols):
        f.add_subplot(rows*2,cols, (2*i*cols)+(j+1))
        plt.imshow(cifar_train_noise[i*cols + j])
        plt.axis("off")

    for j in range(cols):
        f.add_subplot(rows*2,cols,((2*i+1)*cols)+(j+1))
        plt.imshow(cifar_train[i*cols + j])
        plt.axis("off")

f.suptitle("Sample Training Data",fontsize=18)
plt.savefig("Cifar-trian.png")

plt.show()
```



Autoencoders

In [ ]:

```
from keras.layers import Conv2DTranspose, BatchNormalization, add, LeakyReLU
from tensorflow.keras.optimizers import Adam
```

Encoders

In [ ]:

```
inputs = Input(shape=(size,size,channel))

x = Conv2D(32, 3, activation='relu', padding='same')(inputs)
x = BatchNormalization()(x)
x = MaxPool2D()(x)
x = Dropout(0.5)(x)
skip = Conv2D(32, 3, padding='same')(x) # skip connection for decoder
x = LeakyReLU()(skip)
x = BatchNormalization()(x)
x = MaxPool2D()(x)
x = Dropout(0.5)(x)
x = Conv2D(64, 3, activation='relu', padding='same')(x)
x = BatchNormalization()(x)
encoded = MaxPool2D()(x)
```

Decoder

In [ ]:

```
x = Conv2DTranspose(64, 3,activation='relu',strides=(2,2), padding='same')(encoded)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Conv2DTranspose(32, 3, activation='relu',strides=(2,2), padding='same')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Conv2DTranspose(32, 3, padding='same')(x)
x = add([x,skip]) # adding skip connection
x = LeakyReLU()(x)
x = BatchNormalization()(x)
decoded = Conv2DTranspose(3, 3, activation='sigmoid',strides=(2,2), padding='same')(x)
```

In [ ]:

```
autoencoder = Model(inputs, decoded)
autoencoder.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy')
autoencoder.summary()
```

```
Model: "model_1"
__________________________________________________________________________
________________________
 Layer (type)                   Output Shape         Param #     Connected
to
==========================================================================
========================
 input_2 (InputLayer)           [(None, 32, 32, 3)]  0           []

 conv2d_3 (Conv2D)              (None, 32, 32, 32)   896         ['input_2
[0][0]']

 batch_normalization_6 (BatchNo  (None, 32, 32, 32)  128         ['conv2d_
3[0][0]']
 rmalization)

 max_pooling2d_3 (MaxPooling2D)  (None, 16, 16, 32)  0           ['batch_n
ormalization_6[0][0]']

 dropout_4 (Dropout)            (None, 16, 16, 32)   0           ['max_poo
ling2d_3[0][0]']

 conv2d_4 (Conv2D)              (None, 16, 16, 32)   9248        ['dropout
_4[0][0]']

 leaky_re_lu_2 (LeakyReLU)      (None, 16, 16, 32)   0           ['conv2d_
4[0][0]']

 batch_normalization_7 (BatchNo  (None, 16, 16, 32)  128         ['leaky_r
e_lu_2[0][0]']
 rmalization)

 max_pooling2d_4 (MaxPooling2D)  (None, 8, 8, 32)    0           ['batch_n
ormalization_7[0][0]']

 dropout_5 (Dropout)            (None, 8, 8, 32)     0           ['max_poo
ling2d_4[0][0]']

 conv2d_5 (Conv2D)              (None, 8, 8, 64)     18496       ['dropout
_5[0][0]']

 batch_normalization_8 (BatchNo  (None, 8, 8, 64)    256         ['conv2d_
5[0][0]']
 rmalization)

 max_pooling2d_5 (MaxPooling2D)  (None, 4, 4, 64)    0           ['batch_n
ormalization_8[0][0]']

 conv2d_transpose_4 (Conv2DTran  (None, 8, 8, 64)    36928       ['max_poo
ling2d_5[0][0]']
 spose)

 batch_normalization_9 (BatchNo  (None, 8, 8, 64)    256         ['conv2d_
transpose_4[0][0]']
 rmalization)

 dropout_6 (Dropout)            (None, 8, 8, 64)     0           ['batch_n
ormalization_9[0][0]']

 conv2d_transpose_5 (Conv2DTran  (None, 16, 16, 32)  18464       ['dropout
_6[0][0]']
```

```
 spose)

 batch_normalization_10 (BatchN  (None, 16, 16, 32)  128         ['conv2d_
transpose_5[0][0]']
 ormalization)

 dropout_7 (Dropout)            (None, 16, 16, 32)   0           ['batch_n
ormalization_10[0][0]']

 conv2d_transpose_6 (Conv2DTran  (None, 16, 16, 32)  9248        ['dropout
_7[0][0]']
 spose)

 add_1 (Add)                    (None, 16, 16, 32)   0           ['conv2d_
transpose_6[0][0]',
                                                                  'conv2d_

4[0][0]']

 leaky_re_lu_3 (LeakyReLU)      (None, 16, 16, 32)   0           ['add_1
[0][0]']

 batch_normalization_11 (BatchN  (None, 16, 16, 32)  128         ['leaky_r
e_lu_3[0][0]']
 ormalization)

 conv2d_transpose_7 (Conv2DTran  (None, 32, 32, 3)   867         ['batch_n
ormalization_11[0][0]']
 spose)

==========================================================================
========================
Total params: 95,171
Trainable params: 94,659
Non-trainable params: 512
__________________________________________________________________________
________________________
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: Use
rWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Training

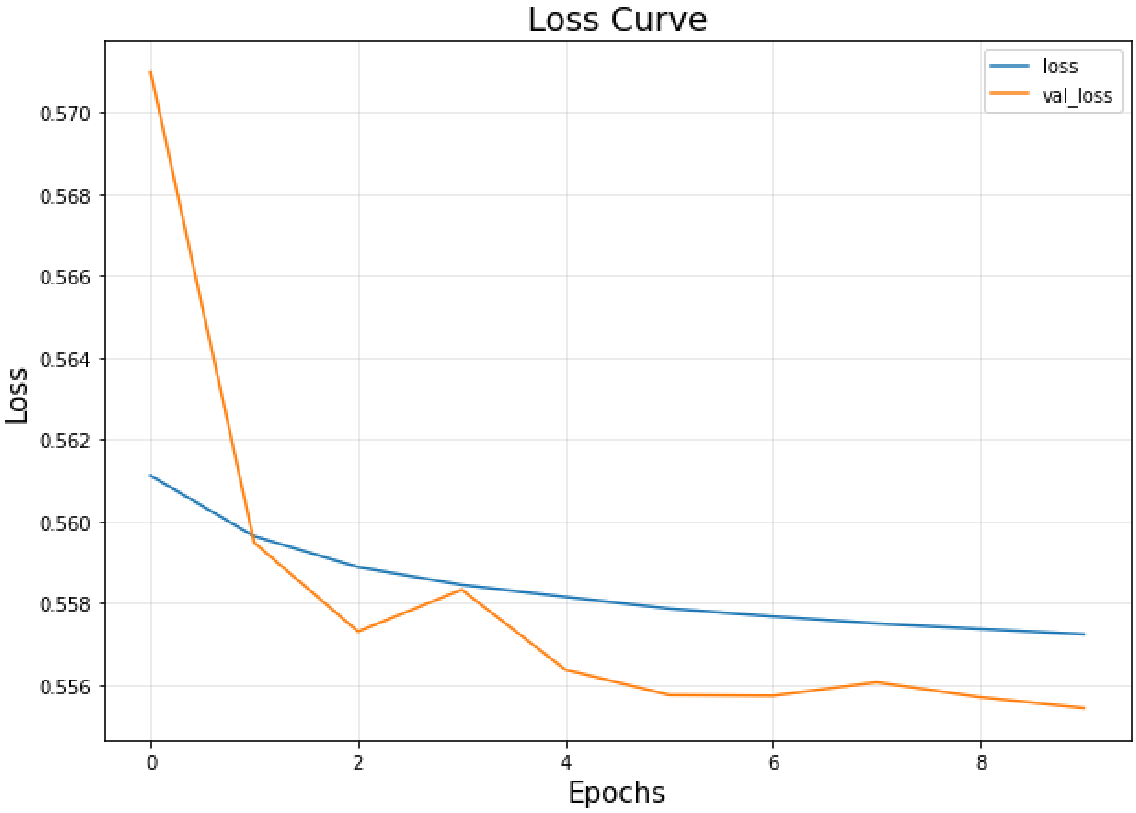
In [ ]:

```
epochs = 10
batch_size = 256

history = autoencoder.fit(cifar_train_noise,
                cifar_train,
                epochs=epochs,
                batch_size=batch_size,
                shuffle=True,
                validation_data=(cifar_test_noise, cifar_test)
               )
```

```
Epoch 1/10
196/196 [==============================] - 160s 815ms/step - loss: 0.5611
- val_loss: 0.5710
Epoch 2/10
196/196 [==============================] - 160s 817ms/step - loss: 0.5596
- val_loss: 0.5595
Epoch 3/10
196/196 [==============================] - 160s 817ms/step - loss: 0.5589
- val_loss: 0.5573
Epoch 4/10
196/196 [==============================] - 160s 815ms/step - loss: 0.5584
- val_loss: 0.5583
Epoch 5/10
196/196 [==============================] - 160s 815ms/step - loss: 0.5581
- val_loss: 0.5564
Epoch 6/10
196/196 [==============================] - 159s 811ms/step - loss: 0.5579
- val_loss: 0.5558
Epoch 7/10
196/196 [==============================] - 159s 812ms/step - loss: 0.5577
- val_loss: 0.5557
Epoch 8/10
196/196 [==============================] - 159s 812ms/step - loss: 0.5575
- val_loss: 0.5561
Epoch 9/10
196/196 [==============================] - 159s 812ms/step - loss: 0.5574
- val_loss: 0.5557
Epoch 10/10
196/196 [==============================] - 160s 816ms/step - loss: 0.5572
- val_loss: 0.5554
```

In [ ]:

```
# Defining Figure
f = plt.figure(figsize=(10,7))
f.add_subplot()

#Adding Subplot
plt.plot(history.epoch, history.history['loss'], label = "loss") # Loss curve for train
ing set
plt.plot(history.epoch, history.history['val_loss'], label = "val_loss") # Loss curve f
or validation set

plt.title("Loss Curve",fontsize=18)
plt.xlabel("Epochs",fontsize=15)
plt.ylabel("Loss",fontsize=15)
plt.grid(alpha=0.3)
plt.legend()
plt.savefig("Loss_curve_cifar10.png")
plt.show()
```



Plotting

In [ ]:

```
num_imgs = 48
rand = np.random.randint(1, cifar_test_noise.shape[0]-48)

cifar_test_images = cifar_test_noise[rand:rand+num_imgs] # slicing
cifar_test_desoided = autoencoder.predict(cifar_test_images) # predict
```

In [ ]:

```
rows = 4 # defining no. of rows in figure
cols = 12 # defining no. of colums in figure
cell_size = 1.5
f = plt.figure(figsize=(cell_size*cols,cell_size*rows*2)) # defining a figure
f.tight_layout()
for i in range(rows):
    for j in range(cols):
        f.add_subplot(rows*2,cols, (2*i*cols)+(j+1)) # adding sub plot to figure on each iteration
        plt.imshow(cifar_test_images[i*cols + j])
        plt.axis("off")

    for j in range(cols):
        f.add_subplot(rows*2,cols,((2*i+1)*cols)+(j+1)) # adding sub plot to figure on each iteration
        plt.imshow(cifar_test_desoided[i*cols + j])
        plt.axis("off")

f.suptitle("Autoencoder Results - Cifar10",fontsize=18)
plt.savefig("test_results_cifar10.png")

plt.show()
```