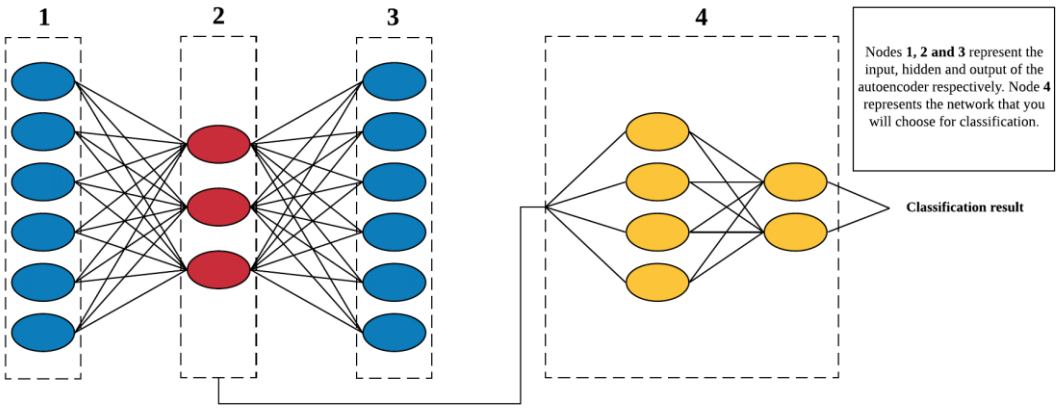
Listing Code

Fadilla Zennifa



In [1]:
```
#loaddata
import sys
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D, Dropout,BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [2]:
```
#make training data 50% of dataset
X_train,y_train = X_train[:int(0.5 * len(X_train))],y_train[:int(0.5 * len(y_train))]
```

In [3]:
```
print('Images Shape: {}'.format(X_train.shape))
print('Labels Shape: {}'.format(y_train.shape))
```

In [4]:
```
#X_train.max()
X_train = X_train/255
X_test = X_test/255
```

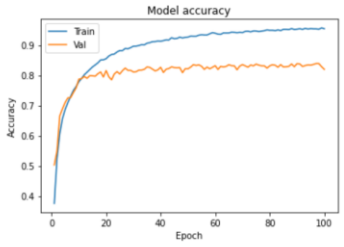
In [5]:
```
#CNN MODEL
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.3))
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

In [6]:
```
#compile model
opt = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy', metrics=['sparse_categorical_accuracy'])
```

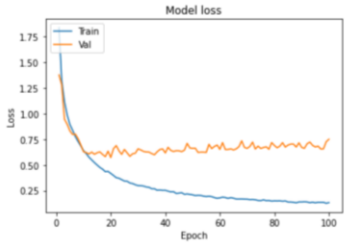
In [7]:
```
#model fitting
history = model.fit(X_train, y_train, epochs=100, verbose=1, validation_data=(X_test, y_test))

#history = model.fit(X_trainnew, y_trainnew, epochs=10,validation_data=(X_testnew, y_testnew))
```

```
Train on 25000 samples, validate on 10000 samples
Epoch 1/100
25000/25000 [==============================] - 312s 12ms/sample - loss: 1.8340 - sparse_categorical_accuracy: 0.3754 - val_loss: 1.3
752 - val_sparse_categorical_accuracy: 0.5026
Epoch 2/100
25000/25000 [==============================] - 347s 14ms/sample - loss: 1.3230 - sparse_categorical_accuracy: 0.5245 - val_loss: 1.2
799 - val_sparse_categorical_accuracy: 0.5492
Epoch 3/100
25000/25000 [==============================] - 346s 14ms/sample - loss: 1.1182 - sparse_categorical_accuracy: 0.6054 - val_loss: 0.9
454 - val_sparse_categorical_accuracy: 0.6652
Epoch 4/100
25000/25000 [==============================] - 331s 13ms/sample - loss: 0.9868 - sparse_categorical_accuracy: 0.6562 - val_loss: 0.8
951 - val_sparse_categorical_accuracy: 0.6888
Epoch 5/100
25000/25000 [==============================] - 345s 14ms/sample - loss: 0.8928 - sparse_categorical_accuracy: 0.6872 - val_loss: 0.8
305 - val_sparse_categorical_accuracy: 0.7102
Epoch 6/100
25000/25000 [==============================] - 336s 13ms/sample - loss: 0.8369 - sparse_categorical_accuracy: 0.7097 - val_loss: 0.7
957 - val_sparse_categorical_accuracy: 0.7250
Epoch 7/100
```

In [8]:
```
# Plot training & validation accuracy values
epoch_range = range(1, 101)
plt.plot(epoch_range, history.history['sparse_categorical_accuracy'])
plt.plot(epoch_range, history.history['val_sparse_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```



In [9]:
```
# Plot training & validation loss values
plt.plot(epoch_range, history.history['loss'])
plt.plot(epoch_range, history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```

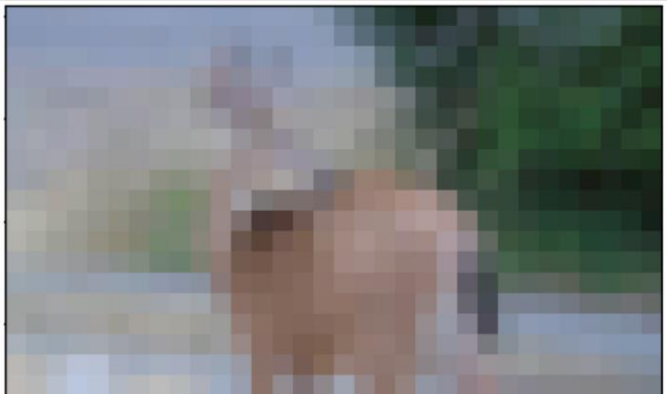


In [10]:
```
# Save the entire model as a SavedModel.
!mkdir -p saved_model
model.save('saved_model/ridgehomedilla')
```

In [12]:
```
new_model = tf.keras.models.load_model('saved_model/ridgehomedilla')
```

In [14]:
```
#evaluate model
from IPython.display import Image
Image("deer.png")
#print('Images Shape: {}'.format(X_train.shape))
```

Out[14]:



In [15]:
```
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)


img = tf.read_file("deer.png")
img = tf.image.decode_jpeg(img, channels=3)
img.set_shape([None, None, 3])
img = tf.image.resize_images(img, (32, 32))
img = img.eval(session=sess) # convert to numpy array
img = np.expand_dims(img, 0) # make 'batch' of 1
# prepare pixel data
img = img.astype('float32')
img = img / 255.0

#pred = model.predict(img)
result = new_model.predict_classes(img)
print(result[0])
#acc

#class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
  #             'dog', 'frog', 'horse', 'ship', 'truck']
```

```
WARNING:tensorflow:From /home/agemono/.pyenv/versions/anaconda3-4.4.0/lib/python3.6/site-packages/tensorflow_core/python/util/tf_should_use.py:198: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.
4
```

In [26]:
```
img = tf.read_file("Aeroplan.jpg")
img = tf.image.decode_jpeg(img, channels=3)
img.set_shape([None, None, 3])
img = tf.image.resize_images(img, (32, 32))
img = img.eval(session=sess) # convert to numpy array
img = np.expand_dims(img, 0) # make 'batch' of 1
# prepare pixel data
img = img.astype('float32')
img = img / 255.0

#pred = model.predict(img)
result = new_model.predict_classes(img)
print(result[0])
Image("Aeroplan.jpg")
```

```
0
```

Out[26]:

In [27]:
```
img = tf.read_file("cat1.jpg")
img = tf.image.decode_jpeg(img, channels=3)
img.set_shape([None, None, 3])
img = tf.image.resize_images(img, (32, 32))
img = img.eval(session=sess) # convert to numpy array
img = np.expand_dims(img, 0) # make 'batch' of 1
# prepare pixel data
img = img.astype('float32')
img = img / 255.0

#pred = model.predict(img)
result = new_model.predict_classes(img)
print(result[0])
Image("cat1.jpg")
```

3

3

Out[27]: