# FINAL PROJECT

DS-5220 Supervised Machine Learning

Sashank Reddy Vasepalli

Khoury College of Computer Sciences

## Summary:

Our goal is to address the classification problem on the CIFAR-10 dataset[1)] using various algorithms. Specifically, we will employ two methods in python using tensorflow & keras: Deep Neural Networks (DNN) with Convolution Layers (CNN) and DNN without Convolution Layers. We will evaluate the performance of each model by analyzing its accuracy and other relevant metrics. Additionally, we will compare the outcomes of the two algorithms to find the strengths and weaknesses of each model.

## Methods:

1. Data Loading:

   The dataset comprises 60,000 32x32 color images categorized into 10 classes, each containing 6,000 images. Of the total images, 50,000 are for training and 10,000 for testing. Specifically, the training batches contain precisely 5,000 images from each class, while the test batch comprises 1,000 randomly chosen images from each class. The CIFAR-10 dataset contains ten distinct classes, which include "airplane," "automobile," "bird," "cat," "deer," "dog," "frog," "horse," "ship," and "truck" respectively.

   We will be loading the CIFAR-10 data using Keras' inbuilt datasets. The command for this is:

   ```
   from keras.datasets import cifar10
   (x_train, y_train), (x_test, y_test) = cifar10.load_data()
   ```

   Once we have loaded and split the CIFAR-10 dataset into training and testing sets, we will begin with the preprocessing steps.

2. Data Preprocessing:

Every image in the CIFAR-10 dataset comprises 32x32 pixels and three colors, Red, Green, and Blue, resulting in a shape of 32x32x3. Each pixel in these images has a value ranging from 0 to 255 for each color. Therefore, our initial step is to normalize these pixel values. Normalizing image data from 0-255 to 0-1 before feeding it to a Neural Network offers several benefits, such as stabilizing the training process, ensuring equal contribution of each feature to the output, and enhancing the model's generalization ability. Following that, we will convert the y_train and y_test outputs from numeric to categorical, which will assist us in making comparisons and generating plots later on. The below commands demonstrate the data normalization process:

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

#one hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

3. CNN Model Architecture and Design Choices:

We will begin by creating a Deep Network with Convolution Layers, often referred to as a Convolutional Neural Network (CNN). The architecture for the CNN model is as follows:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)(32)          (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 activation (Activation)     (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)(32)        (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 activation_1 (Activation)   (None, 32, 32, 32)        0

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D) (64)      (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)
```

```
activation_2 (Activation)    (None, 16, 16, 64)        0

conv2d_3 (Conv2D) (64)       (None, 16, 16, 64)        36928

batch_normalization_3 (Batc  (None, 16, 16, 64)        256
hNormalization)

activation_3 (Activation)    (None, 16, 16, 64)        0

max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)          0
2D)

dropout_1 (Dropout)          (None, 8, 8, 64)          0

conv2d_4 (Conv2D) (128)      (None, 8, 8, 128)         73856

batch_normalization_4 (Batc  (None, 8, 8, 128)         512
hNormalization)

activation_4 (Activation)    (None, 8, 8, 128)         0

conv2d_5 (Conv2D) (128)      (None, 8, 8, 128)         147584

batch_normalization_5 (Batc  (None, 8, 8, 128)         512
hNormalization)

activation_5 (Activation)    (None, 8, 8, 128)         0

global_average_pooling2d (G  (None, 128)               0
lobalAveragePooling2D)

dropout_2 (Dropout)          (None, 128)               0

dense (Dense)                (None, 10)                1290

=================================================================
Total params: 290,090
Trainable params: 289,194
Non-trainable params: 896
_________________________________________________________________
```

We have adopted an architecture similar to VGGNet[2], with 4 major blocks.

First, we have two Conv_2D layers with 32 filters with “same” padding and ReLU activation, followed by a MaxPooling2D layer and Dropout layer (dropout rate = 0.2). We are using 2 Conv_2D layers together so as to allow the network to learn more complex features before reducing the spatial dimensions with pooling. This structure is repeated for 2 more blocks, just with increasing number of filters, 64 & 128. The filter sizes are increased as we go deeper into the network to allow the model to learn a more diverse set of features. In the early layers, the filters capture low-level features such as edges and corners, while the later layers capture high-level

features, like object parts or entire objects. By increasing the number of filters, the model can learn more complex and diverse patterns, which can ultimately improve its performance.

Then, for the final block, we will be using a GlobalAveragePooling2D layer to compute the average value for each feature map, reducing the number of parameters and making the model more resistant to overfitting. The output of the global average pooling layer is directly connected to the final output softmax layer. A softmax layer is used in neural networks for multi-class classification problems to convert the output of the last layer into probabilities. The primary purpose of the softmax layer is to transform the raw scores (logits) produced by the previous layer into a probability distribution that sums to 1, making it suitable for classification tasks.

Additionally, we are using BatchNormalization after each Conv_2D layer and before its activation layer to reach convergence faster, improve regularization to reduce overfitting, reduce sensitivity to weight initializations, and help in training deeper networks by tackling the vanishing gradient problem, which ensures that the gradients do not vanish or explode as they backpropagate through the layers.

4. DNN Model Architecture and Design Choices:

We will then create a Deep Neural Network without using Convolution layers. This is achieved by using Dense layers as seen in the architecture below:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 3072)              0

 dense_1 (Dense)             (None, 2048)              6293504

 batch_normalization_6 (Batc  (None, 2048)             8192
 hNormalization)

 activation_6 (Activation)   (None, 2048)              0

 dropout_3 (Dropout)         (None, 2048)              0

 dense_2 (Dense)             (None, 1024)              2098176

 batch_normalization_7 (Batc  (None, 1024)             4096
 hNormalization)

 activation_7 (Activation)   (None, 1024)              0

 dropout_4 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 512)               524800

 batch_normalization_8 (Batc  (None, 512)              2048
```

```
hNormalization)

activation_8 (Activation)    (None, 512)               0

dropout_5 (Dropout)          (None, 512)               0

dense_4 (Dense)              (None, 256)               131328

batch_normalization_9 (Batc  (None, 256)               1024
hNormalization)

activation_9 (Activation)    (None, 256)               0

dropout_6 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 128)               32896

batch_normalization_10 (Bat  (None, 128)               512
chNormalization)

activation_10 (Activation)   (None, 128)               0

dropout_7 (Dropout)          (None, 128)               0

dense_6 (Dense)              (None, 10)                1290

=================================================================
Total params: 9,097,866
Trainable params: 9,089,930
Non-trainable params: 7,936
_________________________________________________________________
```

This model is divided into 6 blocks: 5 dense layers and 1 softmax output layer. First, we flatten the input data and feed it to the first dense layer of 2048 fully connected neurons. We are using similar techniques as in the CNN model, such as BatchNormalization and ReLU activation. The next 4 dense blocks have the same structure, with reducing neurons: 1024, 512, 256, and 128 respectively. This reducing neuron architecture is known as a "funnel" or "pyramid" architecture and is based on the idea that the initial layers capture low-level features, while the subsequent layers gradually combine these low-level features to form higher-level, more abstract representations. As the model progresses, the spatial dimensions are reduced, and the number of neurons is decreased, allowing the model to focus on the most important features. At the end of the model, we attach a softmax layer block to convert the output of the last layer into probabilities.

5. Model Training:

The above 2 models both use the following as their hyperparameters:

- Optimizer: Adam
- Loss: categorical_crossentropy
- Metrics: accuracy

Adam optimizer is used because of its adaptive learning rate which helps reduce the number of parameters to tune, and its faster convergence speeds which helps us train the model to an optimal point faster.

We are using categorical crossentropy as our Loss function because CIFAR-10 data is of categorical type.

Lastly, we will be using accuracy as our metric of importance.

Additionally, we will be running the model for 300 epochs with a batch size of 100 images. However, we often do not reach the 300$^{th}$ epoch due to the use of an Early Stoppage function. This function allows the model to stop and revert to the best fit model once the monitored parameter no longer improves. Here are the parameters used for but CNN and DNN models:

*CNN Model*

*overfitCallback = keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta = 0.0025, patience = 30, restore_best_weights=True)*

*cnn = cnn_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=300, batch_size = 100, callbacks=[overfitCallback])*

*DNN Model*

*overfitCallback = keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta = 0.005, patience = 30, restore_best_weights=True)*

*dnn = dnn_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=300, batch_size = 100, callbacks=[overfitCallback])*

# Results:

Convolutional Neural Network:

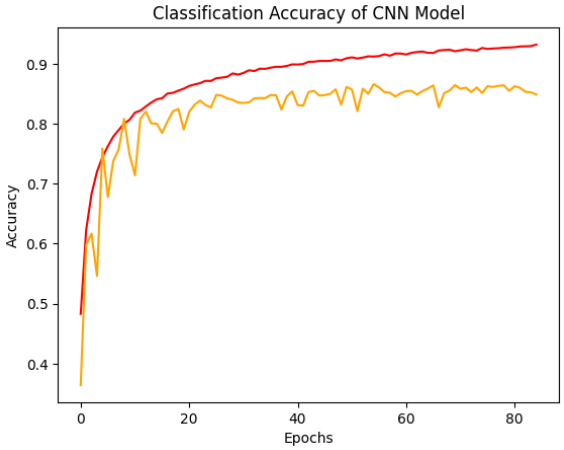
The CNN model trained for a total of 85/300 Epochs, before the Early Stoppage function was triggered. Each epoch took 8 seconds to run. The best model was from Epoch 55/300, with the following statistics:

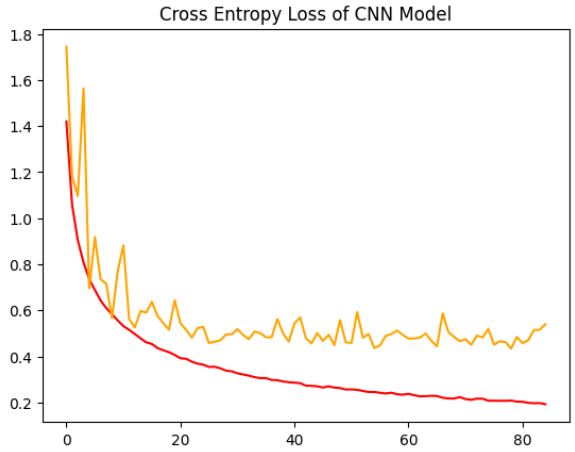
```
Epoch 55/300
500/500 [==============================] - 8s 17ms/step - loss: 0.24
56 - accuracy: 0.9123 - val_loss: 0.4363 - val_accuracy: 0.8663
```

The test accuracy of the model after using the model.evaluate function was:

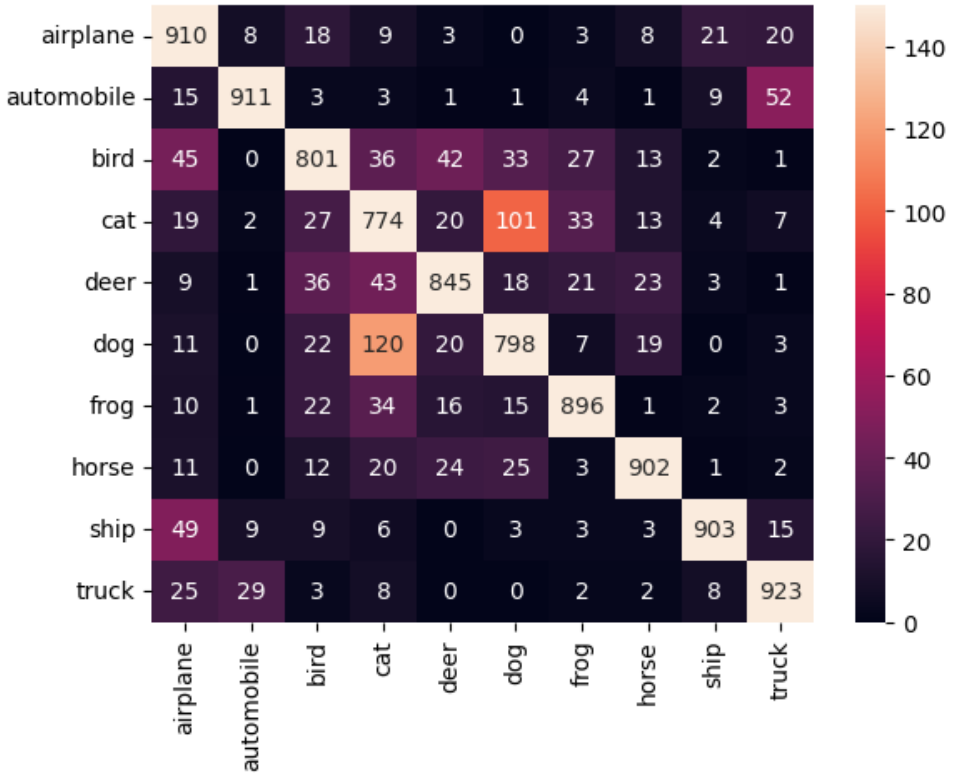
```
Test Accuracy:  0.8662999868392944
```

Below are the graphs for Accuracy and Cross Entropy Loss (Red -> Train | Orange -> Test):




As we see from the above graphs, both test Accuracy and test Cross Entropy Loss stop improving beyond 30 epochs. But the train accuracy and error keep improving. This is a sign of Model Saturation, which means the model has converged, is starting to overfit and cannot learn any more new features from the current data. The confusion matrix of the different class predictions is given below:



From the confusion matrix, we can make the following observations:

- Dog and Cat are often misclassified.
- Living creatures like birds, cats, deer, dogs, frogs, and horses are more likely to be misclassified as other living creatures.
- Inanimate objects like Airplanes, automobiles, ships, and trucks are more likely to be misclassified as other inanimate objects.

## Deep Neural Network without Convolution layers:

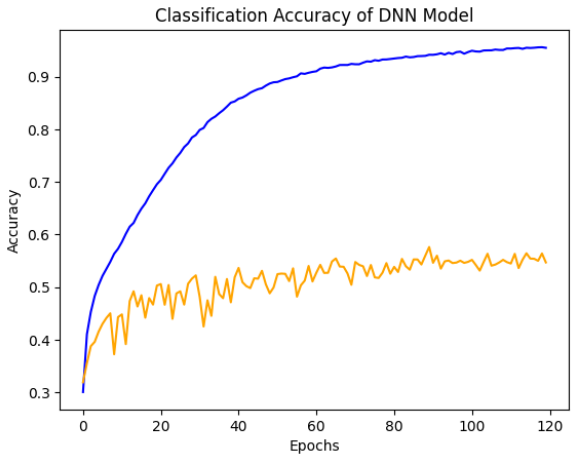
The DNN model trained for a total of 120/300 Epochs, before the Early Stoppage function was triggered. Each epoch took 4 seconds to run. The best model was from Epoch 90/300, with the following statistics:

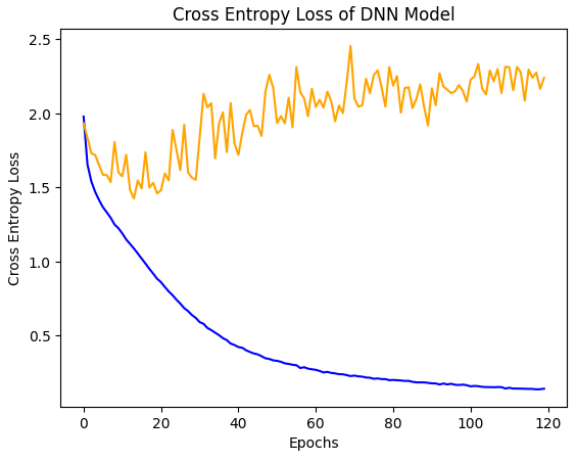
```
Epoch 90/300
500/500 [==============================] - 4s 8ms/step - loss: 0.180
7 - accuracy: 0.9417 - val_loss: 1.9187 - val_accuracy: 0.5762
```

The test accuracy of the model after using the model.evaluate function was:

```
Test Accuracy:  0.576200008392334
```

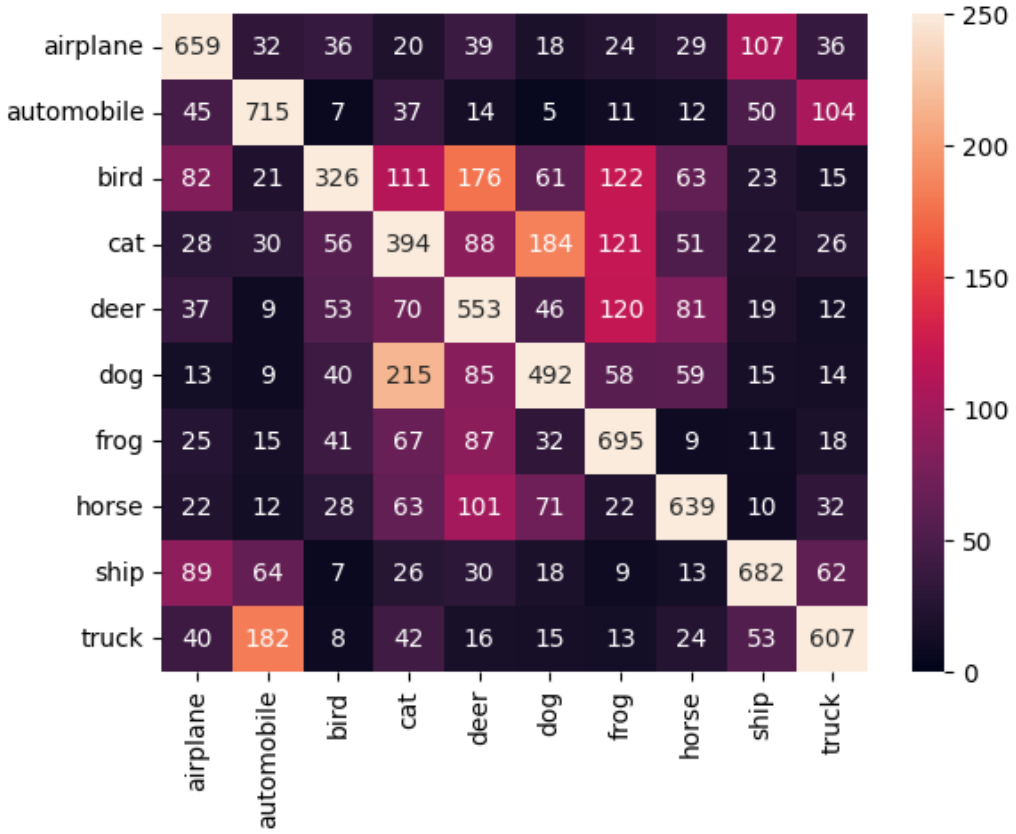
Below are the graphs for Accuracy and Cross Entropy Loss (Blue -> Train | Orange -> Test):




From the figures shown above, it appears that the test accuracy gradually increases as more epochs are completed. However, the test Cross Entropy Loss reaches a peak at 20 epochs and has been increasing ever since. This suggests that although the model is making more accurate predictions, the loss from incorrect predictions has also increased significantly. In this situation, the model can be fine-tuned to meet specific requirements. For instance, I have trained the model to prioritize generalization to new data, which is why I have terminated the training process despite the model's accuracy continuing to improve.

The confusion matrix of the different class predictions is given below:




From the confusion matrix, we can make the following observations:

- Dog and Cat are often misclassified.
- A lot of birds have been misclassified as cats, deer, and frogs.
- Living creatures like birds, cats, deer, dogs, frogs, and horses are more likely to be misclassified as other living creatures.
- Inanimate objects like Airplanes, automobiles, ships, and trucks are more likely to be misclassified as other inanimate objects.

## Comparison of Models:

| Convolutional Neural Network | Deep Neural Network |
|---|---|
| Total Training time till convergence:<br>8 seconds/epoch * 85 epochs = 680 Seconds | Total Training time till convergence:<br>4 seconds/epoch * 120 epochs = 480 Seconds |
| No. of parameters:<br>290,090 | No. of parameters:<br>9,097,866 |
| Validation Loss:<br>0.4363 | Validation Loss:<br>1.9187 |
| Validation Accuracy:<br>86.63% | Validation Accuracy:<br>57.62% |

We can summarize that the CNN model is more accurate with an accuracy of over 86% compared to the 57% of the DNN. Also, the misclassifications are significantly lesser for the CNN model with a better-looking confusion matrix as well as lower validation loss of 0.4363 compared to the 1.9187 validation loss of the DNN.

Hence, despite taking longer to train while using vastly lesser parameters, a Convolutional Neural Network performs significantly better at classification with the CIFAR-10 dataset when compared to a Deep Neural Network without Convolution layers.

CNNs work better than DNNs without convolution layers for image classification because they are specifically designed to handle the spatial structure of image data. A DNN cannot take advantage of the spatial structure since it treats each pixel independently and does not capture the spatial relationships between them. Meanwhile, combination of convolutional layers and pooling layers in CNNs enables them to learn hierarchical representations of image data and capture the spatial structure of images, resulting in better performance for image classification tasks compared to traditional DNNs without convolutional layers.

## References:


1) Official CIFAR-10 Website https://www.cs.toronto.edu/~kriz/cifar.html
2) VGGNet architecture https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6
3) Github Repository https://github.com/sashank3/SML_Final_project


## Code:

```
# In[1]:

import keras

from keras.datasets import cifar10

from keras.utils import to_categorical

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix

import tensorflow as tf

from tensorflow.keras import models, layers


# In[2]:

num_classes = 10


#loading data

(x_train, y_train), (x_test, y_test) = cifar10.load_data()


# Each pixel has value 0-255, hence by dividing by 255 we attain a value between 0 and 1.

x_train, x_test = x_train / 255.0, x_test / 255.0


#one hot encoding

y_train = to_categorical(y_train)

y_test = to_categorical(y_test)
```

```
# In[3]:
#CNN Model Architecture:

cnn_model = models.Sequential([
    layers.Conv2D(32, (3, 3), padding='same', input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Conv2D(32, (3, 3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(rate = 0.2),

    layers.Conv2D(64, (3, 3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Conv2D(64, (3, 3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(rate = 0.3),

    layers.Conv2D(128, (3, 3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.Conv2D(128, (3, 3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
```

```
    layers.GlobalAveragePooling2D(),
    keras.layers.Dropout(rate = 0.4),

    layers.Dense(num_classes, activation = 'softmax')
])

# In[4]:
cnn_model.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy'])

cnn_model.summary()



# In[5]:
#Early stoppage callback function
overfitCallback = keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta = 0.0025, patience = 30, restore_best_weights=True)

#Model training
cnn = cnn_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=300, batch_size = 100, callbacks=[overfitCallback])



# In[6]:
#CNN model Classification Accuracy

plt.title('Classification Accuracy of CNN Model')
plt.plot(cnn.history['accuracy'], color='red', label='train')
```

```
plt.plot(cnn.history['val_accuracy'], color='orange', label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")



# In[7]:
#CNN model Cross Entropy Loss

plt.title('Cross Entropy Loss of CNN Model')
plt.plot(cnn.history['loss'], color='red', label='train')
plt.plot(cnn.history['val_loss'], color='orange', label='test')
plt.xlabel("Epochs")
plt.ylabel("Cross Entropy Loss")



# In[8]:
#Using best fit model from training to predict on test data
y_pred_cnn =(cnn_model.predict(x_test))

#Calculation of Accuracy of model
cnn_loss, cnn_accuracy = cnn_model.evaluate(x_test, y_test)
print("Test Accuracy: ", cnn_accuracy)



# In[9]:
#Confusion matrix plot
cf_matrix = confusion_matrix(y_test.argmax(1),y_pred_cnn.argmax(1))

#Seaborn confusion matrix visualization
```

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

sns.heatmap(cf_matrix, annot=True, fmt="d",  xticklabels = class_names, yticklabels = class_names, vmin=0, vmax=150)

#############################################################################

# In[11]:

#Deep Neural Network Architecture without convolutions:

dnn_model = models.Sequential([

    layers.Flatten(input_shape=(32, 32, 3)),

    layers.Dense(2048),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    keras.layers.Dropout(rate=0.3),

    layers.Dense(1024),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    keras.layers.Dropout(rate=0.2),

    layers.Dense(512),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    keras.layers.Dropout(rate=0.3),

    layers.Dense(256),
```

```
    layers.BatchNormalization(),
    layers.Activation('relu'),
    keras.layers.Dropout(rate=0.4),

    layers.Dense(128),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    keras.layers.Dropout(rate=0.5),

    layers.Dense(num_classes, activation='softmax')
])



# In[12]:

dnn_model.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy'])

dnn_model.summary()



# In[13]:

#Early stoppage callback function

overfitCallback = keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta = 0.005,
patience = 30, restore_best_weights=True)

#Model training

dnn = dnn_model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=300, batch_size =
100, callbacks=[overfitCallback])
```

```
# In[14]:
#DNN model Classification Accuracy

plt.title('Classification Accuracy of DNN Model')
plt.plot(dnn.history['accuracy'], color='blue', label='train')
plt.plot(dnn.history['val_accuracy'], color='orange', label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")


# In[15]:
#DNN model Cross Entropy Loss

plt.title('Cross Entropy Loss of DNN Model')
plt.plot(dnn.history['loss'], color='blue', label='train')
plt.plot(dnn.history['val_loss'], color='orange', label='test')
plt.xlabel("Epochs")
plt.ylabel("Cross Entropy Loss")


# In[16]:
#Using best fit model from training to predict on test data
y_pred_dnn =(dnn_model.predict(x_test))

#Calculation of Accuracy of model
dnn_loss, dnn_accuracy = dnn_model.evaluate(x_test, y_test)
print("Test Accuracy: ", dnn_accuracy)
```

```
# In[17]:

#Confusion matrix plot

cf_matrix = confusion_matrix(y_test.argmax(1),y_pred_dnn.argmax(1))


#Seaborn confusion matrix visualization

class_names = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]

sns.heatmap(cf_matrix, annot=True, fmt="d",  xticklabels = class_names, yticklabels =
class_names, vmin=0, vmax=250)
```