IMAGE CLASSIFICATION USING THE CIFAR-10 DATASET

Authors: Indhuja Muthu Kumar, Venkat Srinivasa Raghavan

1. OBJECTIVE:

The CIFAR-10 dataset is a widely used benchmark dataset for image classification tasks. It consists of 60,000 images, with 50,000 images used for training and 10,000 images used for testing. Each image is a 32x32 pixel RGB image, with 10 possible labels corresponding to the 10 different object classes. The objective of this project is to build four different classification models to accurately classify the images into their respective classes. Specifically, Support Vector Machines (SVM), Random Forest Classifier, Deep Neural Networks (DNN) with Convolutional Neural Networks (CNN), and DNN without CNN were utilized for classification. The performance of each model was evaluated using standard evaluation metrics, such as accuracy, precision, recall, and F1-score, on the test set. Additionally, the convergence speed of the models during training will be tracked by measuring the time taken to reach a certain accuracy level.

2. METHODS

2.1. CONVOLUTION NEURAL NETWORK:

A convolutional neural network (CNN) is a type of artificial neural network designed to process and analyze images. It is composed of one or more convolutional layers, which apply a set of filters (also known as kernels) to the input data. The filters convolve across the input data, performing a mathematical operation that highlights certain features or patterns in the data.

***2.1.1 Data Preprocessing*:** The initial stage of any machine learning analysis involves data preprocessing. To begin with, the CIFAR-10 dataset was imported, and the images were transformed into a format that is compatible with machine learning algorithms. Additionally, the pixel values were normalized to enhance the speed of convergence of the algorithms.

2.1.2 Feature Extraction: The Convolutional layers aid in extracting features from the images. Specifically, a pre-trained VGG16 model was used to extract features from the images, and a fully connected neural network was trained on top of the extracted features. Furthermore, the most used pooling technique, max pooling, was utilized in the model. It also aids in extracting the most important and relevant information from the previous convolutional layer, which makes the network more efficient and robust.

***2.1.3 Model Building*:** Upon extracting feature, the model is built by introducing activation functions. Since convolution is a linear operation that may not effectively capture the nonlinear relationships in image data, non-linear activation functions such as the Rectified Linear Unit (ReLU) has been to the model applied after convolutional layers to introduce nonlinearity into the activation maps. After training, the pre-trained VGG16 model was fine-tuned using SGD with Cross-entropy loss function.

2.1.4 Evaluation**:** The performance of the trained model was then evaluated on the test set based on the accuracy metrics like precision, recall, and F-1 score. The test loss for each epoch was also tracked and an overall accuracy of 81% was obtained on the test data.

2.2 DNN without CNN

A Deep Neural Network (DNN) is a type of neural network that can be used to process a variety of data types, including images. Unlike CNN, DNN is not specialized for image processing and does not use convolutional or pooling layers. Instead, a DNN typically consists of fully connected layers that are stacked on top of each other.

2.2.1 Data Preprocessing**:** Once the CIFAR-10 dataset was imported, the images were then transformed into a format that is compatible with DNN. Additionally, the pixel values were normalized to enhance the speed of convergence of the algorithm and to mitigate issues arising from weight matrix initialization.

2.2.2 Feature Extraction**:** Unlike DNN with CNN, the DNN model doesn't include Convolution layers to extract appropriate features. Once the features are engineered, they can be fed into the DNN as input data. The DNN then learns to map the input features to the desired output through a series of layers that perform nonlinear transformations on the input data.

2.2.3 Model Building**:** Upon extracting feature, the model is built by introducing activation functions. To achieve this, the model has utilized non-linear activation functions such as the Rectified Linear Unit (ReLU) after the convolutional layers. This aids the network to capture more complex relationships in the data by introducing non-linear transformations. After training, the fully connected layer was fine-tuned using SGD with Cross-entropy loss function.

2.2.4 Evaluation**:** To evaluate the performance of the trained model, accuracy metrics such as precision, recall, and F-1 score were used to assess its performance on the test set. Additionally, the test loss for each epoch was monitored, and the final model achieved an overall accuracy of 45% on the test data.

2.3. RANDOM FOREST CLASSIFIER

The Random Forest algorithm is an ensemble technique that employs multiple decision trees to construct a stronger and more precise model. The principle behind random forest is to create a forest of decision trees, where each tree is trained on a random subset of the training data (bootstrapped data) and a random subset of features. Once the trees are constructed, they each independently classify the data, and the algorithm combines their predictions to obtain a final prediction.

2.3.1 Data Preprocessing**:** The process of data preprocessing for the CIFAR-10 dataset involved utilizing the "load_data" function available within the Keras library. This function was utilized to load the dataset and subsequently split it into both training and testing sets. To enable compatibility with the random forest classifier, the 3D image arrays were flattened to 2D. Further, to ensure consistency and comparability, the pixel values were normalized by dividing them by 255.

2.3.2 Modeling**:** In the modeling stage, the random forest classifier is constructed by initializing the "RandomForestClassifier" class from the “sklearn.ensemble” library. The training of the classifier is then executed by fitting it with the training data. Next, the classifier randomly selects a subset of features and bootstraps the samples to distribute the data across each decision tree within the forest. The final step involves predicting the test data by utilizing the "predict" function.

2.3.3 Evaluation**:** The evaluation of the model is done by calculating the accuracy. The confusion matrix is also plotted to find the number of true and false positives and negatives predicted. Finally, the precision, recall and f1-score of the predictions are found using the classification report.

2.4. SUPPORT VECTOR MACHINE

Support Vector Machines (SVMs) are a class of machine learning algorithms that can be employed for classification tasks. One of the key strengths of SVMs is their capacity to learn intricate associations between features and labels, even in cases where the data is not linearly separable.

2.4.1 Data Preprocessing**:** To begin with, the data is loaded into the model by utilizing the "load_dataset" function available within the Keras library. Once the CIFAR-10 dataset was imported the images were flattened from a 32x32x32 tensor into a 1024-dimensional vector. This has been done to make the data more manageable for the SVM classifier. The data is then split into training and test data.

2.4.2 Feature Extraction**:** In the feature extraction stage, dimensionality reduction was performed through Principal Component Analysis (PCA). The purpose of this technique is to reduce the dimensionality of the data while preserving as much information as possible. The primary benefit of this step is to improve the computational efficiency of the SVM model. The PCA was conducted using the "PCA" function from the ‘sklearn.decomposition’ library, where the number of principal components was determined to be 100. This approach enables the identification of the most salient features of the dataset while minimizing the effects of noise and redundancy in the data.

2.4.3 Model Building**:** In the model building stage, a non-linear SVM classifier was created by initializing it with a 'rbf' kernel. The regularization parameter C was set to 10, and a random state of 42 was specified. To combine the PCA and SVM in a single process, a pipeline was constructed. The pipeline is a sequential set of operations where the output of one step is used as the input for the next. In this case, the pipeline included PCA as a preprocessing step followed by SVM. The pipeline was then fitted with the training data, with an additional scaling step performed using 'StandardScaler'. Finally, predictions were made on the test data using the trained model.

2.4.4 Evaluation: The trained model's effectiveness is evaluated by measuring its accuracy. To gain further insight into the model's performance, a confusion matrix is generated, which displays the number of true and false positive and negative classifications. Additionally, the precision, recall, and f1-score of the model output are calculated using the classification report. These evaluation metrics provide a comprehensive overview of the model's ability to correctly classify new, unseen data.

3. RESULTS:

3.1 Preliminary analysis results: Before data modeling, the data was analyzed to check whether there were any imbalances in the class distribution. The analysis showed the there was no class imbalance. The same can be seen in the figure below.

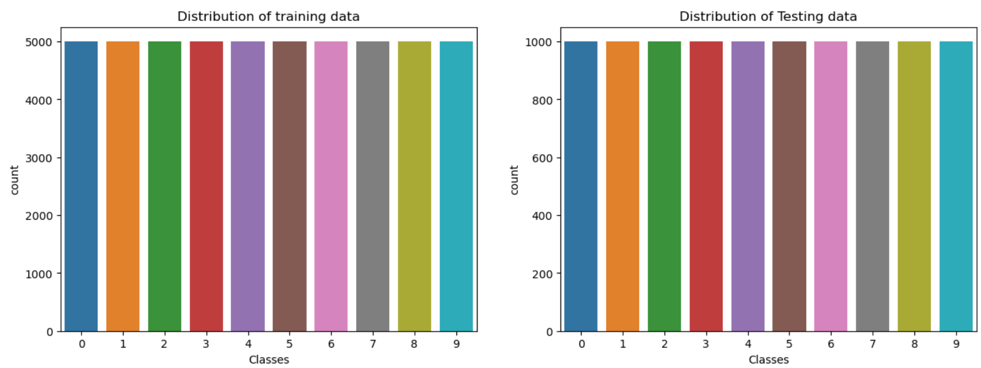


Figure 1: Class distribution in the training and test data

Images belonging to different categories in the dataset were also visualized as shown in figure 2.

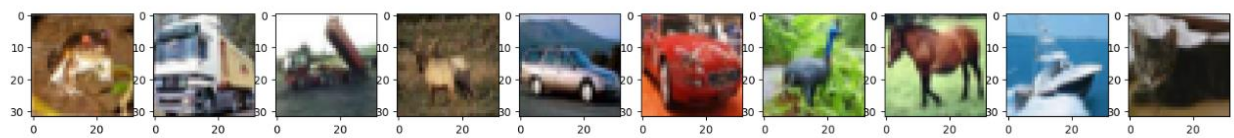


Figure 2: Images of different classes of the output variable

DNN with CNN: The DNN model with CNN achieved a test accuracy of 0.81, meaning that 81% of the test data was correctly classified. The test loss was 0.605, indicating the difference between predicted and actual labels was relatively low. The change in the model's loss and accuracy for every epoch is shown in figure 3.

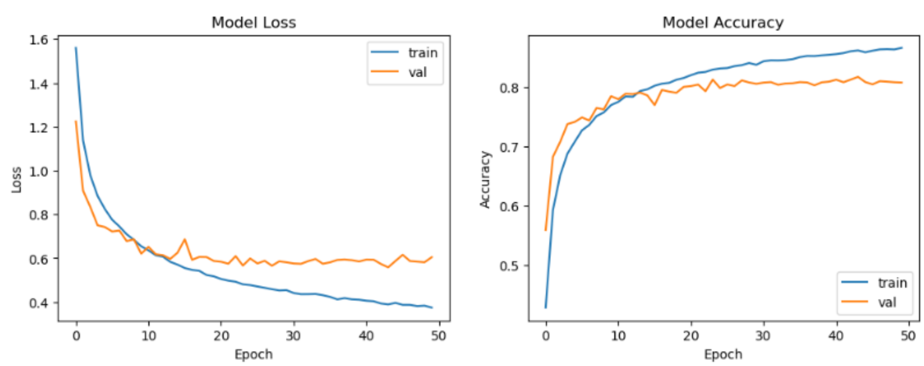


Figure 3: Loss and accuracy per epoch for the DNN model with CNN

The precision of the model was 0.81, which represents the proportion of true positive classifications in relation to all positive classifications. The recall was also 0.81, which represents the proportion of true positive classifications in relation to all actual positive samples. The f1 score was 0.81, which is the harmonic mean of precision and recall. To summarize these values, a classification report has been printed out as shown in figure 4.

	precision	recall	f1-score	support
0	0.87	0.80	0.83	1000
1	0.90	0.90	0.90	1000
2	0.81	0.65	0.72	1000
3	0.63	0.68	0.66	1000
4	0.77	0.79	0.78	1000
5	0.77	0.71	0.74	1000
6	0.76	0.92	0.83	1000
7	0.89	0.82	0.86	1000
8	0.87	0.91	0.89	1000
9	0.84	0.91	0.87	1000
accuracy			0.81	10000
macro avg	0.81	0.81	0.81	10000
weighted avg	0.81	0.81	0.81	10000

Figure 4: Classification report for the DNN model with CNN

To visualize the number of true and false positive and negative classifications in this classification task, the confusion matrix has been plotted as shown in figure 5.

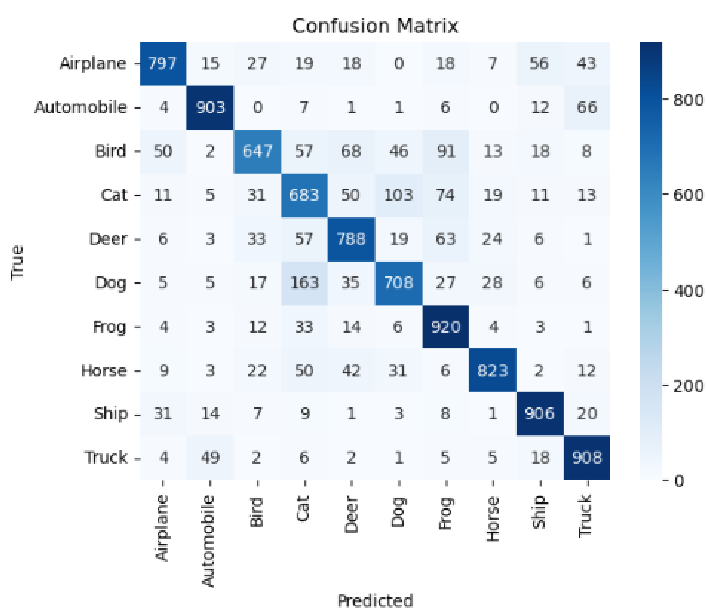


Figure 5: Confusion matrix for the Deep Neural network model with CNN.

3.2 DNN without CNN: The DNN model without CNN achieved a test accuracy of 0.4425, meaning that only 44.25% of the test data was correctly classified. The test loss was 1.6543, indicating a relatively high difference between predicted and actual labels. The variation in the model's loss and accuracy with each epoch can be seen in figure 6.

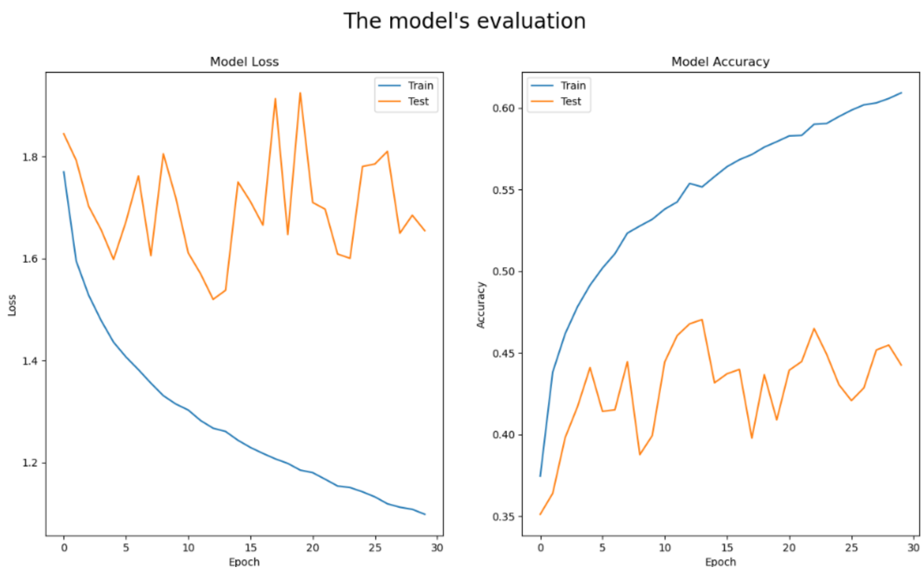


Figure 6: Loss and accuracy per epoch for the DNN model without CNN

The precision of the model was 0.47, which represents the proportion of true positive classifications in relation to all positive classifications. The recall was 0.45, which represents the proportion of true positive classifications in relation to all actual positive samples. The F1-Score was 0.45, which is the harmonic mean of precision and recall. To summarize these values, a classification report has been printed out as shown in figure 7.

	precision	recall	f1-score	support
0	0.43	0.64	0.51	1000
1	0.56	0.51	0.54	1000
2	0.37	0.19	0.25	1000
3	0.30	0.29	0.29	1000
4	0.44	0.19	0.27	1000
5	0.35	0.51	0.41	1000
6	0.48	0.54	0.51	1000
7	0.50	0.51	0.51	1000
8	0.49	0.66	0.56	1000
9	0.55	0.39	0.46	1000
accuracy			0.44	10000
macro avg	0.45	0.44	0.43	10000
weighted avg	0.45	0.44	0.43	10000

Figure 7: Classification report for the DNN model without CNN

To visualize the number of true and false positive and negative classifications in this classification task, the confusion matrix has been plotted as shown in figure 8.

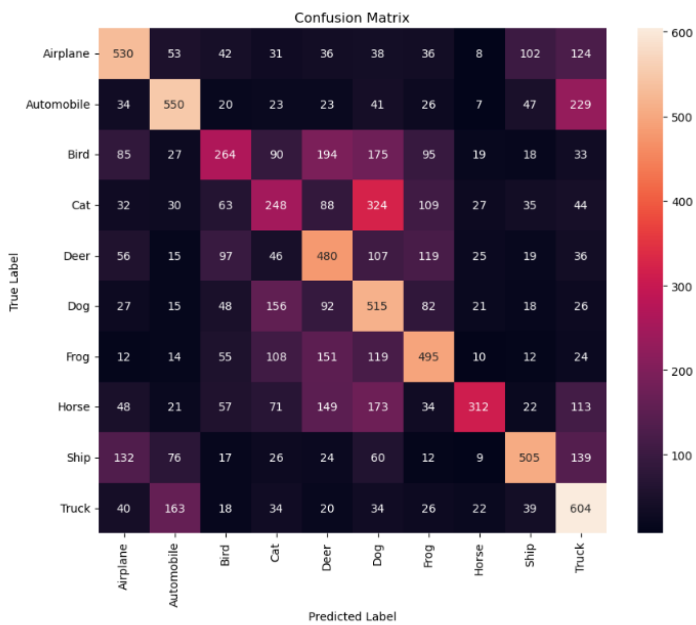


Figure 8: Confusion matrix for the DNN model without CNN

3.3 Random Forest: The random forest model achieved a test accuracy of 0.4654, meaning that only 46.54% of the test data was correctly classified. The precision of the model was 0.46, which represents the proportion of true positive classifications in relation to all positive classifications. The recall was 0.47, which represents the proportion of true positive classifications in relation to all actual positive samples. The f1 score was 0.46, which is the harmonic mean of precision and recall. To summarize these values, a classification report has been printed out as shown in figure 9.

Classification Report:

	precision	recall	f1-score	support
0	0.54	0.56	0.55	1000
1	0.52	0.54	0.53	1000
2	0.38	0.33	0.35	1000
3	0.33	0.28	0.30	1000
4	0.39	0.38	0.39	1000
5	0.43	0.40	0.41	1000
6	0.47	0.57	0.52	1000
7	0.51	0.45	0.48	1000
8	0.58	0.61	0.59	1000
9	0.47	0.52	0.50	1000
accuracy			0.47	10000
macro avg	0.46	0.47	0.46	10000
weighted avg	0.46	0.47	0.46	10000

Figure 9: Classification report for the Random Forest classifier model

To visualize the number of true and false positive and negative classifications in this classification task, the confusion matrix has been plotted as shown in figure 10.

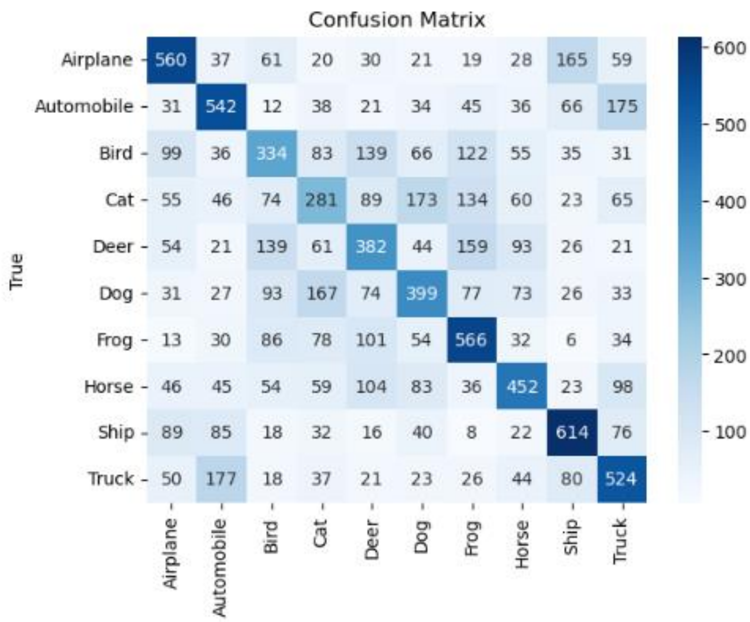


Figure 10: Confusion matrix for the Random Forest classifier model

Support Vector Machine: The SVM model achieved a test accuracy of 0.55, meaning that only 55% of the test data was correctly classified. The precision of the model was 0.55, which represents the proportion of true positive classifications in relation to all positive classifications. The recall was also 0.55, which represents the proportion of true positive classifications in relation to all actual positive samples. The f1 score was 0.55, which is the harmonic mean of precision and recall. To summarize these values, a classification report has been printed out as shown in figure 11.

	precision	recall	f1-score	support
0	0.60	0.64	0.62	1000
1	0.64	0.68	0.66	1000
2	0.43	0.44	0.44	1000
3	0.35	0.36	0.36	1000
4	0.48	0.49	0.49	1000
5	0.48	0.44	0.46	1000
6	0.59	0.62	0.61	1000
7	0.65	0.58	0.62	1000
8	0.67	0.66	0.66	1000
9	0.62	0.58	0.60	1000
accuracy			0.55	10000
macro avg	0.55	0.55	0.55	10000
weighted avg	0.55	0.55	0.55	10000

Figure 11: Classification report for the SVM Classifier with rbf Kernel

To visualize the number of true and false positive and negative classifications in this classification task, the confusion matrix has been plotted as shown in figure 12.

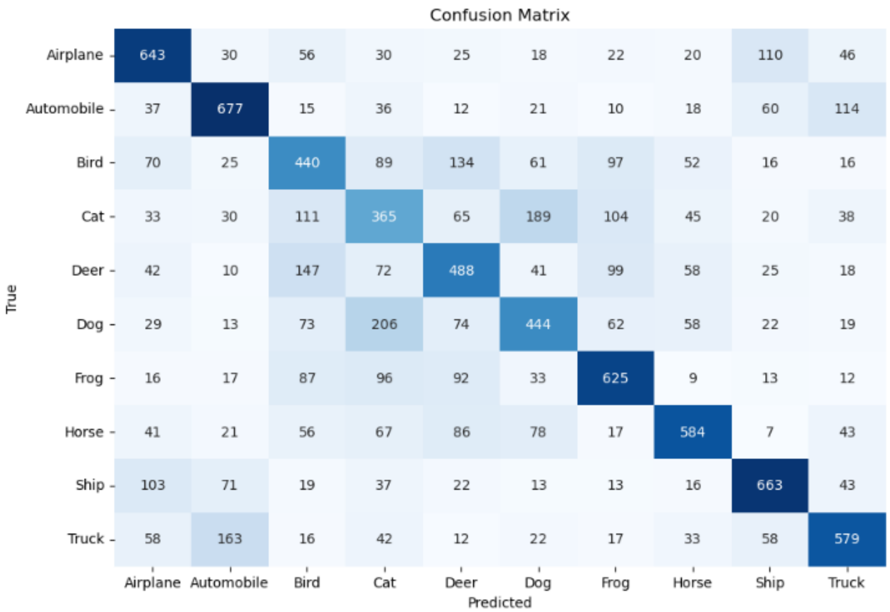


Figure 12: Confusion matrix for classification with SVM with non-linear 'rbf' Kernel

4. COMPARISON

The convergence time of each model is as follows:

- DNN with CNN: 364.07 seconds for 50 epochs
- DNN without CNN: 70.6456 seconds for 30 epochs
- Random Forest: 234.1155 seconds
- SVM: 318.452 seconds

From these metrics, it can be observed that the DNN with CNN takes significantly longer than the other models to converge, as it requires 50 epochs to train and there are multiple convolution layers. The SVM takes longer than the Random Forest, but shorter than the DNN with CNN. The DNN without CNN has the shortest convergence time of all the models.

However, it is important to note that convergence time alone cannot be used to determine the effectiveness of a model. Other factors such as accuracy, precision, recall, and f1 score should also be considered to conclude the performance of a model. The summary of the performance metrics is given in the table below.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
DNN With CNN	**81%**	**81%**	**81%**	**80.7%**
DNN Without CNN	44.25%	45%	44%	45%
SVM	55%	55%	55%	55%
Random Forest	46.54%	46%	47%	46%

In terms of accuracy, the DNN with CNN achieved the highest performance with a test accuracy of 0.81 followed by SVM with an accuracy of 0.55, while DNN without CNN and Random Forest had lower accuracy at 0.4425 and 0.4654, respectively. When looking at precision, recall, and f1 score, the DNN with CNN outperformed the other models with scores of 0.81, 0.81, and 0.807, respectively. SVM performed slightly better than Random Forest in terms of precision, recall, and f1 score with scores of 0.55 for all three metrics, while Random Forest achieved scores of 0.46, 0.47, and 0.46, respectively.

Overall, the DNN with CNN outperformed the other models in terms of accuracy and precision, recall, and f1 score. SVM also showed reasonable performance with an accuracy of 0.55 and similar scores for precision, recall, and f1 score. However, the DNN without CNN and Random Forest had lower performance in all metrics. Therefore, based on these results, the DNN with CNN can be considered the better model for CIFAR-10 dataset image classification.

5. INFERENCE

Comparing the performance of all the 4 models, the Deep Neural Network with CNN model achieved an impressive test accuracy of 0.81, which means that it correctly classified 81% of the images in the test set. The test loss of 0.605 indicates that the model's predictions were on average, very close to the true labels. Additionally, the precision, recall, and f1-score for the model were all 0.81, which suggests that the model was effective at identifying both true positives and true negatives, with a balance between precision and recall. Thus, the high-test accuracy achieved by the Deep Neural Network with CNN model, as well as its high precision, recall, and f1-score, demonstrate that it was able to accurately classify images across all classes, making it the best model for CIFAR-10 image classification.

6. CODE

The code implementation for the above project can be found in the subsequent pages.

Deep Neural Network Method with Convolution

The classification deep learning model has been created with convolution layers. The data has been split into training and test data. The convolution layers are inserted using the Conv2D function with ReLU as the activation function. Max Pooling has also been done to adjust the size of the image. Dropout has also been implemented to prevent overfitting and regularize the model. The architectureuses the adam optimizer algorithm. The metrics for evaluation of each epoch is accuracy. There are a total of 2 hidden layers and the output layer consists of a softmax classifier. The change in accuracy and the loss are visualized for gaining insights.

In [ ]:

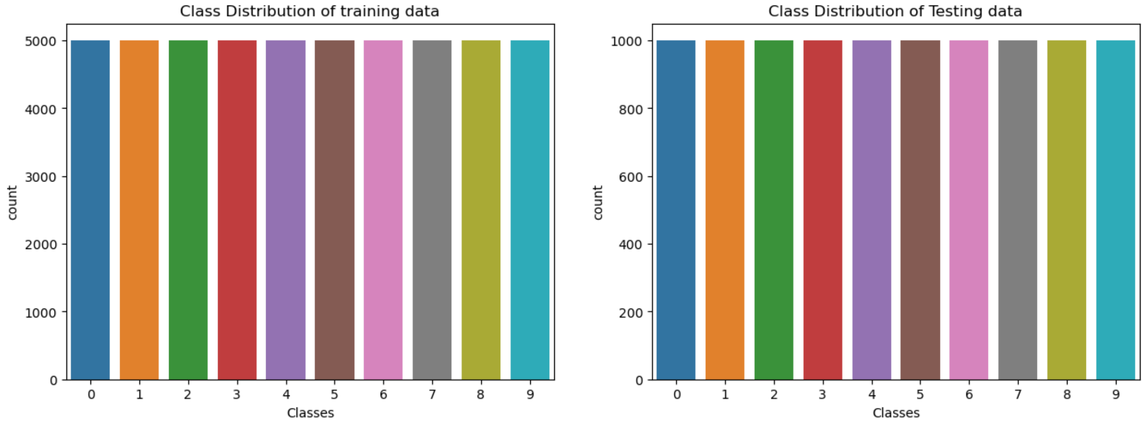
```
#importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.utils import to_categorical
from keras.datasets import cifar10
from sklearn.metrics import accuracy_score
import pandas as pd
from keras import datasets
from keras.utils import np_utils
import seaborn as sns
from keras import layers, models
from sklearn.metrics import accuracy_score, precision_score, recall_score
import warnings
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
# Disable all warnings
import time
warnings.filterwarnings("ignore")
```

In [ ]:

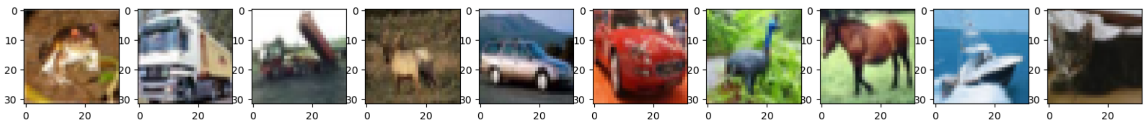
```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

fig, axs = plt.subplots(1,2,figsize=(15,5))
# Count plot for training set
sns.countplot(x = y_train.ravel(), ax=axs[0])
axs[0].set_title('Class Distribution of training data')
axs[0].set_xlabel('Classes')
# Count plot for testing set
sns.countplot(x = y_test.ravel(), ax=axs[1])
axs[1].set_title('Class Distribution of Testing data')
axs[1].set_xlabel('Classes')
plt.show()
# Convert class labels to one-hot encoded vectors
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

```python
# Normalize pixel values to be between 0 and 1
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
Classes = ['Airplane', 'Automobile','Bird','Cat','Deer','Dog','Frog','Hor
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.ta
r.gz
170498071/170498071 [==============================] - 3s 0us/step
```



In [ ]:
```python
fig, axes = plt.subplots(1,10,figsize=(20,10))
for i in range(0,10):
    axes[i].imshow(x_train[i])
plt.show()
```



In [ ]:
```python
# Define model architecture
model = Sequential()
#Adding Convolution layer with relu activation
model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_sha
model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
#Adding a softmax classifier layer for multicalss classification
model.add(Dense(num_classes, activation='softmax'))
```

In [ ]:
```python
# Compile the model
start_time = time.time()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=

# Train the model
history = model.fit(x_train, y_train, batch_size=64, epochs=50, validatio
end_time = time.time()
# Evaluate the model
scores = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
print('Convergence time:', end_time - start_time)
```

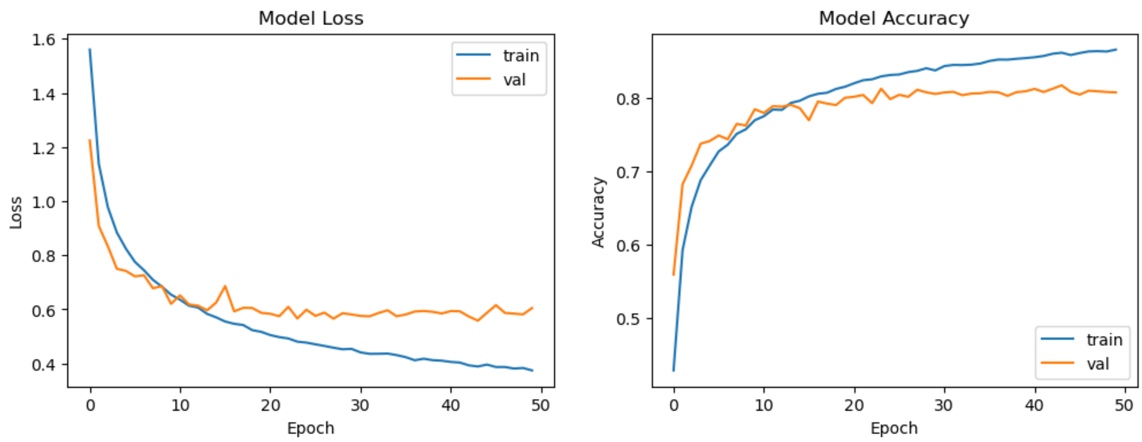
```
Epoch 1/50
2023-04-24 18:10:52.750909: E tensorflow/core/grappler/optimizers/meta_op
timizer.cc:954] layout failed: INVALID_ARGUMENT: Size of values 0 does no
t match size of permutation 4 @ fanin shape insequential/dropout/dropout/
SelectV2-2-TransposeNHWCToNCHW-LayoutOptimizer
782/782 [==============================] - 19s 10ms/step - loss: 1.5602 -
accuracy: 0.4284 - val_loss: 1.2245 - val_accuracy: 0.5593
Epoch 2/50
782/782 [==============================] - 7s 9ms/step - loss: 1.1376 - a
ccuracy: 0.5930 - val_loss: 0.9080 - val_accuracy: 0.6829
Epoch 3/50
782/782 [==============================] - 7s 9ms/step - loss: 0.9781 - a
ccuracy: 0.6519 - val_loss: 0.8335 - val_accuracy: 0.7082
Epoch 4/50
782/782 [==============================] - 7s 9ms/step - loss: 0.8841 - a
ccuracy: 0.6883 - val_loss: 0.7500 - val_accuracy: 0.7382
Epoch 5/50
782/782 [==============================] - 7s 9ms/step - loss: 0.8244 - a
ccuracy: 0.7083 - val_loss: 0.7422 - val_accuracy: 0.7418
Epoch 6/50
782/782 [==============================] - 7s 9ms/step - loss: 0.7757 - a
ccuracy: 0.7276 - val_loss: 0.7219 - val_accuracy: 0.7494
Epoch 7/50
782/782 [==============================] - 7s 9ms/step - loss: 0.7449 - a
ccuracy: 0.7368 - val_loss: 0.7262 - val_accuracy: 0.7442
Epoch 8/50
782/782 [==============================] - 7s 9ms/step - loss: 0.7092 - a
ccuracy: 0.7517 - val_loss: 0.6780 - val_accuracy: 0.7653
Epoch 9/50
782/782 [==============================] - 7s 9ms/step - loss: 0.6837 - a
ccuracy: 0.7578 - val_loss: 0.6861 - val_accuracy: 0.7630
Epoch 10/50
782/782 [==============================] - 7s 9ms/step - loss: 0.6545 - a
ccuracy: 0.7703 - val_loss: 0.6205 - val_accuracy: 0.7853
Epoch 11/50
782/782 [==============================] - 7s 9ms/step - loss: 0.6358 - a
ccuracy: 0.7757 - val_loss: 0.6516 - val_accuracy: 0.7802
Epoch 12/50
782/782 [==============================] - 7s 9ms/step - loss: 0.6135 - a
ccuracy: 0.7848 - val_loss: 0.6179 - val_accuracy: 0.7894
Epoch 13/50
782/782 [==============================] - 7s 9ms/step - loss: 0.6068 - a
ccuracy: 0.7843 - val_loss: 0.6135 - val_accuracy: 0.7888
Epoch 14/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5834 - a
ccuracy: 0.7939 - val_loss: 0.5966 - val_accuracy: 0.7913
Epoch 15/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5706 - a
ccuracy: 0.7969 - val_loss: 0.6251 - val_accuracy: 0.7866
Epoch 16/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5551 - a
ccuracy: 0.8029 - val_loss: 0.6862 - val_accuracy: 0.7702
Epoch 17/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5467 - a
ccuracy: 0.8064 - val_loss: 0.5925 - val_accuracy: 0.7958
Epoch 18/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5422 - a
ccuracy: 0.8078 - val_loss: 0.6057 - val_accuracy: 0.7930
Epoch 19/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5233 - a
ccuracy: 0.8130 - val_loss: 0.6051 - val_accuracy: 0.7910
Epoch 20/50
782/782 [==============================] - 8s 10ms/step - loss: 0.5169 -
```

```
accuracy: 0.8160 - val_loss: 0.5872 - val_accuracy: 0.8010
Epoch 21/50
782/782 [==============================] - 7s 9ms/step - loss: 0.5050 - a
ccuracy: 0.8207 - val_loss: 0.5837 - val_accuracy: 0.8023
Epoch 22/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4974 - a
ccuracy: 0.8249 - val_loss: 0.5748 - val_accuracy: 0.8048
Epoch 23/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4925 - a
ccuracy: 0.8262 - val_loss: 0.6094 - val_accuracy: 0.7936
Epoch 24/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4806 - a
ccuracy: 0.8301 - val_loss: 0.5662 - val_accuracy: 0.8133
Epoch 25/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4772 - a
ccuracy: 0.8320 - val_loss: 0.5987 - val_accuracy: 0.7991
Epoch 26/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4708 - a
ccuracy: 0.8328 - val_loss: 0.5760 - val_accuracy: 0.8050
Epoch 27/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4648 - a
ccuracy: 0.8360 - val_loss: 0.5880 - val_accuracy: 0.8023
Epoch 28/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4584 - a
ccuracy: 0.8377 - val_loss: 0.5656 - val_accuracy: 0.8119
Epoch 29/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4527 - a
ccuracy: 0.8413 - val_loss: 0.5856 - val_accuracy: 0.8084
Epoch 30/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4540 - a
ccuracy: 0.8383 - val_loss: 0.5812 - val_accuracy: 0.8062
Epoch 31/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4410 - a
ccuracy: 0.8443 - val_loss: 0.5758 - val_accuracy: 0.8082
Epoch 32/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4357 - a
ccuracy: 0.8458 - val_loss: 0.5743 - val_accuracy: 0.8090
Epoch 33/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4358 - a
ccuracy: 0.8457 - val_loss: 0.5867 - val_accuracy: 0.8045
Epoch 34/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4367 - a
ccuracy: 0.8462 - val_loss: 0.5965 - val_accuracy: 0.8067
Epoch 35/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4310 - a
ccuracy: 0.8478 - val_loss: 0.5746 - val_accuracy: 0.8071
Epoch 36/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4235 - a
ccuracy: 0.8512 - val_loss: 0.5811 - val_accuracy: 0.8090
Epoch 37/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4120 - a
ccuracy: 0.8532 - val_loss: 0.5921 - val_accuracy: 0.8084
Epoch 38/50
782/782 [==============================] - 8s 10ms/step - loss: 0.4176 -
accuracy: 0.8530 - val_loss: 0.5941 - val_accuracy: 0.8036
Epoch 39/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4122 - a
ccuracy: 0.8542 - val_loss: 0.5910 - val_accuracy: 0.8086
Epoch 40/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4104 - a
ccuracy: 0.8552 - val_loss: 0.5850 - val_accuracy: 0.8098
Epoch 41/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4058 - a
ccuracy: 0.8563 - val_loss: 0.5937 - val_accuracy: 0.8131
```

```
Epoch 42/50
782/782 [==============================] - 7s 9ms/step - loss: 0.4034 - a
ccuracy: 0.8581 - val_loss: 0.5930 - val_accuracy: 0.8088
Epoch 43/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3931 - a
ccuracy: 0.8611 - val_loss: 0.5736 - val_accuracy: 0.8133
Epoch 44/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3889 - a
ccuracy: 0.8625 - val_loss: 0.5582 - val_accuracy: 0.8179
Epoch 45/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3959 - a
ccuracy: 0.8594 - val_loss: 0.5873 - val_accuracy: 0.8091
Epoch 46/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3870 - a
ccuracy: 0.8621 - val_loss: 0.6155 - val_accuracy: 0.8053
Epoch 47/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3870 - a
ccuracy: 0.8643 - val_loss: 0.5869 - val_accuracy: 0.8106
Epoch 48/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3813 - a
ccuracy: 0.8647 - val_loss: 0.5843 - val_accuracy: 0.8098
Epoch 49/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3834 - a
ccuracy: 0.8642 - val_loss: 0.5814 - val_accuracy: 0.8088
Epoch 50/50
782/782 [==============================] - 7s 9ms/step - loss: 0.3744 - a
ccuracy: 0.8668 - val_loss: 0.6047 - val_accuracy: 0.8083
Test loss: 0.6046910881996155
Test accuracy: 0.8083000183105469
Convergence time: 364.0762469768524
```

In [ ]:

```
# Plot training loss and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```



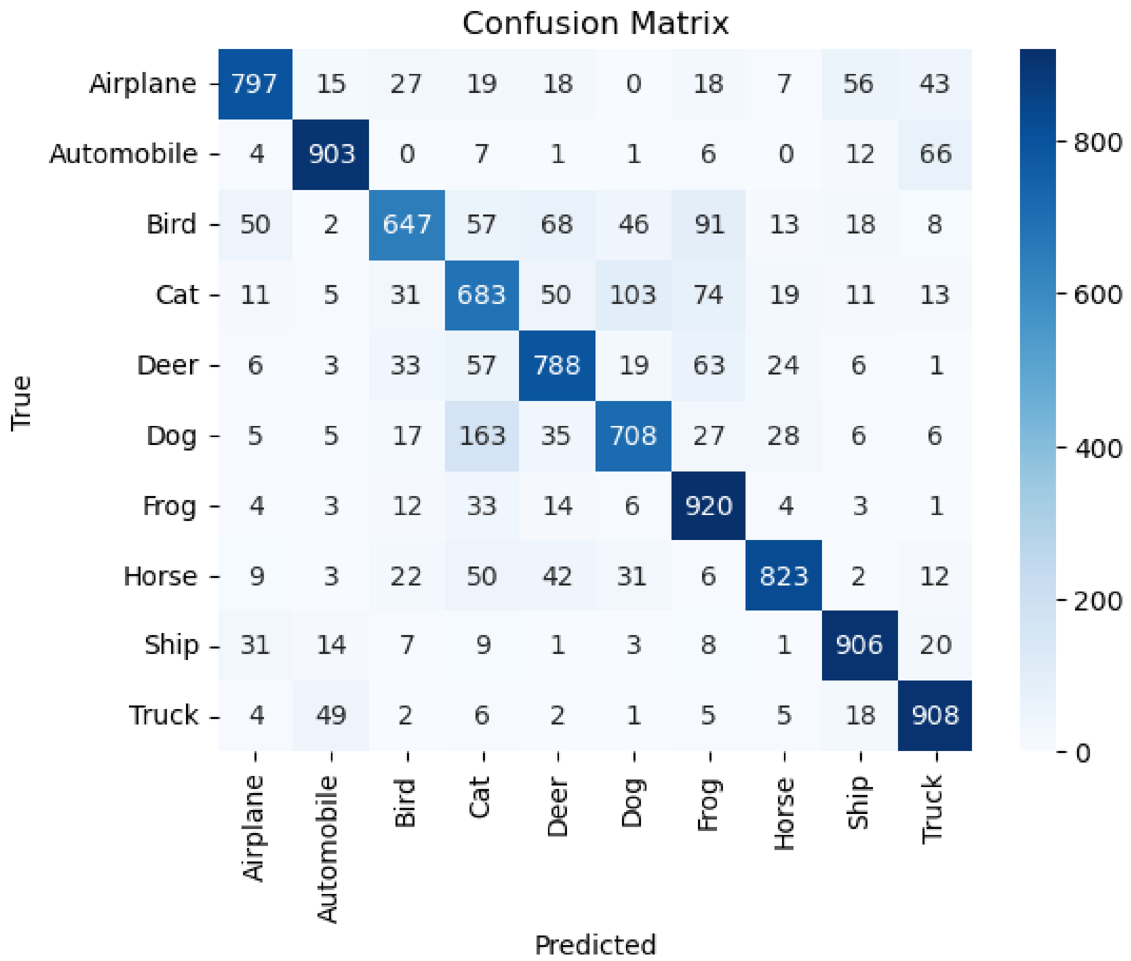
In [ ]:

```
# Make predictions on test data
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)
# Calculate accuracy on test data
accuracy = accuracy_score(y_test_classes, y_pred_classes)
print('Test accuracy:', accuracy)
```

```
313/313 [==============================] - 1s 3ms/step
Test accuracy: 0.8083
```

In [ ]:

```
cm = confusion_matrix(y_test_classes, y_pred_classes)

# Create a heatmap of the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d",xticklabels = Classes,
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:

```
accuracy = accuracy_score(y_test_classes, y_pred_classes)
precision = precision_score(y_test_classes, y_pred_classes, average='macr
recall = recall_score(y_test_classes, y_pred_classes, average='macro')
f1 = f1_score(y_test_classes, y_pred_classes, average='macro')

# Print results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Accuracy: 0.8083
Precision: 0.8113123261533104
```

```
Recall: 0.8083
F1-Score: 0.8073996076679244
```

In [ ]:
```
print(classification_report(y_test_classes, y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.87      0.80      0.83      1000
           1       0.90      0.90      0.90      1000
           2       0.81      0.65      0.72      1000
           3       0.63      0.68      0.66      1000
           4       0.77      0.79      0.78      1000
           5       0.77      0.71      0.74      1000
           6       0.76      0.92      0.83      1000
           7       0.89      0.82      0.86      1000
           8       0.87      0.91      0.89      1000
           9       0.84      0.91      0.87      1000

    accuracy                           0.81     10000
   macro avg       0.81      0.81      0.81     10000
weighted avg       0.81      0.81      0.81     10000
```

Deep Neural Network without convolution

The classification deep learning model has been creataed without convolution layers. The data has been split into training and test data. The shape of the images are found out using the shape parameter. The neural network architecture is then created with ReLU function as the activation function with the adam optimizer algorithm. The metrics for evaluation of each epoch is accuracy. There are a total of 2 hidden layers and the output layer consists of a softmax classifier. The change in accuracy and the loss are visualized for gaining insights.

In [ ]:
```
import numpy as np
from keras.utils import np_utils
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Loading the dataset and splitting it into train and test sets
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Converting labels to one-hot encoded vectors
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)

# Reshaping the input data
L, W, H, C = X_train.shape
X_train = X_train.reshape(-1, W*H*C)
X_test = X_test.reshape(-1, W*H*C)

# Normalizing the input data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

In [ ]:

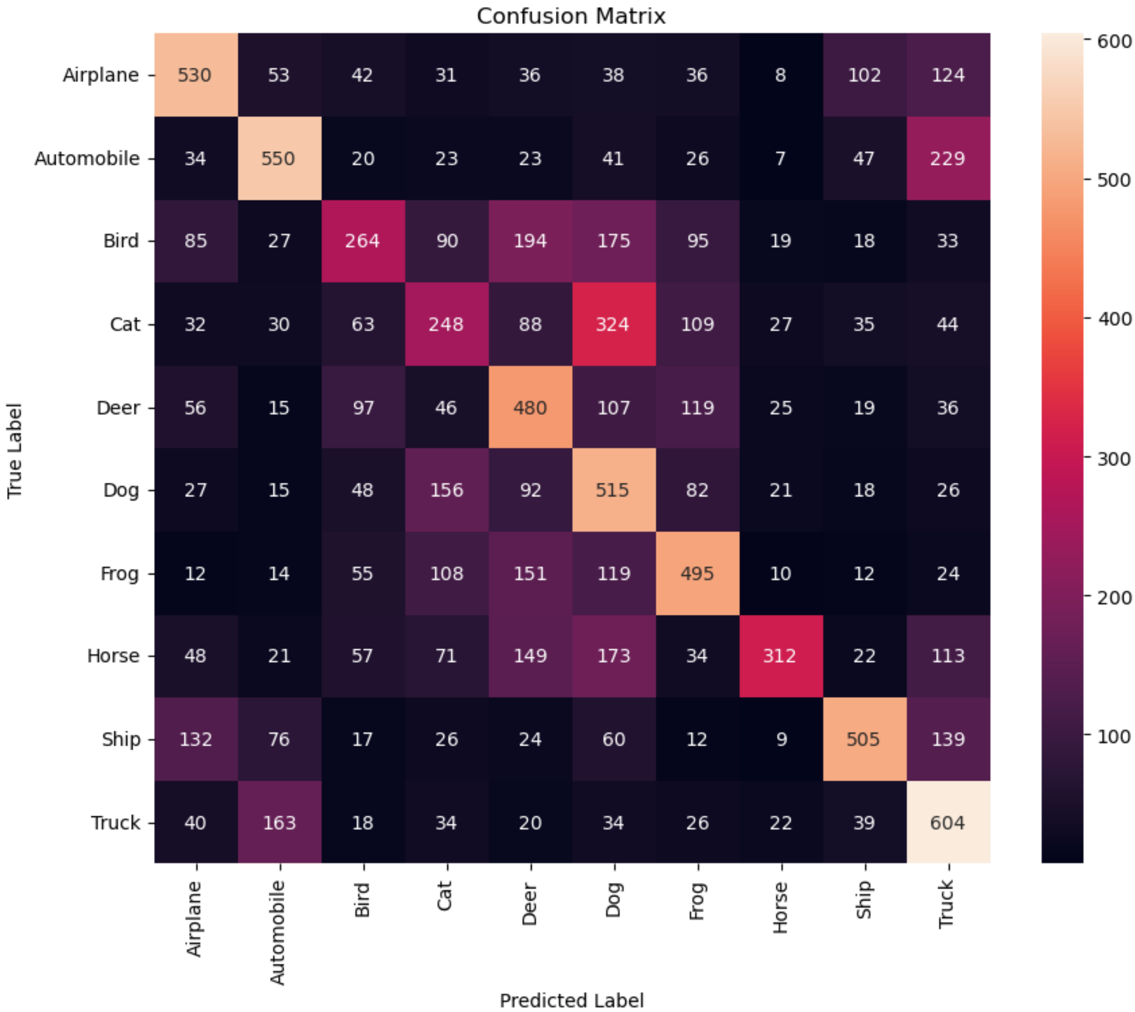
```
# Building the model
model = Sequential()
model.add(Dense(100, input_shape=X_train[1].shape, activation='relu', nam
model.add(BatchNormalization())  # Add batch normalization layer
model.add(Dropout(0))
model.add(Dense(50, activation='relu',name='Hidden-2'))
model.add(BatchNormalization())  # Add batch normalization layer
model.add(Dropout(0))
model.add(Dense(10, activation='softmax'))

# Compiling the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
start_time = time.time()
# Training the model
history = model.fit(X_train, y_train, epochs=30, batch_size=100, validati

# Predicting the labels for test data
y_pred = model.predict(X_test)

# Converting predicted probabilities to class labels
y_pred_classes = np.argmax(y_pred, axis=1)

# Converting one-hot encoded labels to class labels
y_true_classes = np.argmax(y_test, axis=1)
end_time = time.time()
# Calculating confusion matrix
confusion_mtx = confusion_matrix(y_true_classes, y_pred_classes)
```

```
Epoch 1/30
400/400 [==============================] - 5s 6ms/step - loss: 1.7696 - a
ccuracy: 0.3744 - val_loss: 1.8443 - val_accuracy: 0.3511
Epoch 2/30
400/400 [==============================] - 2s 5ms/step - loss: 1.5947 - a
ccuracy: 0.4381 - val_loss: 1.7922 - val_accuracy: 0.3639
Epoch 3/30
400/400 [==============================] - 2s 5ms/step - loss: 1.5278 - a
ccuracy: 0.4616 - val_loss: 1.7022 - val_accuracy: 0.3980
Epoch 4/30
400/400 [==============================] - 2s 5ms/step - loss: 1.4782 - a
ccuracy: 0.4783 - val_loss: 1.6561 - val_accuracy: 0.4171
Epoch 5/30
400/400 [==============================] - 2s 5ms/step - loss: 1.4356 - a
ccuracy: 0.4914 - val_loss: 1.5978 - val_accuracy: 0.4409
Epoch 6/30
400/400 [==============================] - 2s 5ms/step - loss: 1.4065 - a
ccuracy: 0.5019 - val_loss: 1.6721 - val_accuracy: 0.4141
Epoch 7/30
400/400 [==============================] - 2s 5ms/step - loss: 1.3816 - a
ccuracy: 0.5106 - val_loss: 1.7616 - val_accuracy: 0.4151
Epoch 8/30
400/400 [==============================] - 3s 7ms/step - loss: 1.3552 - a
ccuracy: 0.5232 - val_loss: 1.6053 - val_accuracy: 0.4445
Epoch 9/30
400/400 [==============================] - 2s 5ms/step - loss: 1.3306 - a
ccuracy: 0.5276 - val_loss: 1.8052 - val_accuracy: 0.3876
Epoch 10/30
400/400 [==============================] - 2s 5ms/step - loss: 1.3143 - a
ccuracy: 0.5317 - val_loss: 1.7190 - val_accuracy: 0.3993
Epoch 11/30
400/400 [==============================] - 2s 5ms/step - loss: 1.3025 - a
ccuracy: 0.5379 - val_loss: 1.6107 - val_accuracy: 0.4444
Epoch 12/30
400/400 [==============================] - 2s 5ms/step - loss: 1.2817 - a
ccuracy: 0.5423 - val_loss: 1.5694 - val_accuracy: 0.4605
```

```
Epoch 13/30
400/400 [==============================] - 2s 5ms/step - loss: 1.2668 - a
ccuracy: 0.5537 - val_loss: 1.5195 - val_accuracy: 0.4677
Epoch 14/30
400/400 [==============================] - 3s 7ms/step - loss: 1.2604 - a
ccuracy: 0.5515 - val_loss: 1.5373 - val_accuracy: 0.4703
Epoch 15/30
400/400 [==============================] - 2s 5ms/step - loss: 1.2431 - a
ccuracy: 0.5579 - val_loss: 1.7496 - val_accuracy: 0.4316
Epoch 16/30
400/400 [==============================] - 2s 5ms/step - loss: 1.2290 - a
ccuracy: 0.5639 - val_loss: 1.7112 - val_accuracy: 0.4371
Epoch 17/30
400/400 [==============================] - 2s 5ms/step - loss: 1.2174 - a
ccuracy: 0.5682 - val_loss: 1.6652 - val_accuracy: 0.4398
Epoch 18/30
400/400 [==============================] - 2s 5ms/step - loss: 1.2067 - a
ccuracy: 0.5714 - val_loss: 1.9132 - val_accuracy: 0.3977
Epoch 19/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1977 - a
ccuracy: 0.5759 - val_loss: 1.6467 - val_accuracy: 0.4366
Epoch 20/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1846 - a
ccuracy: 0.5792 - val_loss: 1.9245 - val_accuracy: 0.4089
Epoch 21/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1796 - a
ccuracy: 0.5828 - val_loss: 1.7095 - val_accuracy: 0.4393
Epoch 22/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1667 - a
ccuracy: 0.5831 - val_loss: 1.6964 - val_accuracy: 0.4446
Epoch 23/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1534 - a
ccuracy: 0.5899 - val_loss: 1.6084 - val_accuracy: 0.4648
Epoch 24/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1505 - a
ccuracy: 0.5903 - val_loss: 1.6000 - val_accuracy: 0.4492
Epoch 25/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1421 - a
ccuracy: 0.5946 - val_loss: 1.7801 - val_accuracy: 0.4304
Epoch 26/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1321 - a
ccuracy: 0.5985 - val_loss: 1.7851 - val_accuracy: 0.4207
Epoch 27/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1186 - a
ccuracy: 0.6018 - val_loss: 1.8097 - val_accuracy: 0.4285
Epoch 28/30
400/400 [==============================] - 2s 5ms/step - loss: 1.1118 - a
ccuracy: 0.6030 - val_loss: 1.6492 - val_accuracy: 0.4517
Epoch 29/30
400/400 [==============================] - 3s 7ms/step - loss: 1.1077 - a
ccuracy: 0.6057 - val_loss: 1.6845 - val_accuracy: 0.4547
Epoch 30/30
400/400 [==============================] - 2s 5ms/step - loss: 1.0980 - a
ccuracy: 0.6091 - val_loss: 1.6543 - val_accuracy: 0.4425
313/313 [==============================] - 1s 2ms/step
```

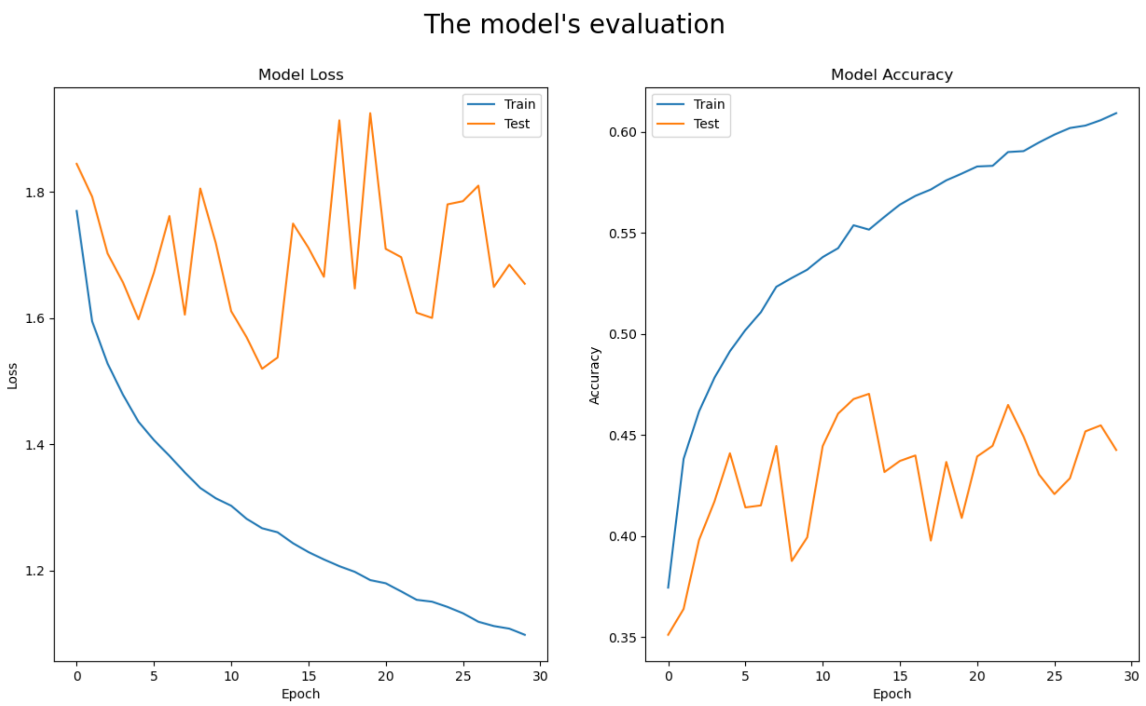
In [ ]:

```python
# Visualizing confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, annot=True, fmt="d", xticklabels = Classes, yt
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:

```
fig,axes = plt.subplots(1,2, figsize=(15,8))
fig.suptitle("The model's evaluation ",fontsize=20)
axes[0].plot(history.history['loss'])
axes[0].plot(history.history['val_loss'])
axes[0].set_title('Model Loss')
axes[0].set_ylabel('Loss')
axes[0].set_xlabel('Epoch')
axes[0].legend(['Train','Test'])

axes[1].plot(history.history['accuracy'])
axes[1].plot(history.history['val_accuracy'])
axes[1].set_title('Model Accuracy')
axes[1].set_ylabel('Accuracy')
axes[1].set_xlabel('Epoch')
axes[1].legend(['Train','Test'])
plt.show()
performance_test = model.evaluate(X_test, y_test, batch_size=100)
pred = model.predict(X_test)
```




```
100/100 [==============================] - 0s 3ms/step - loss: 1.6389 - a
ccuracy: 0.4503
313/313 [==============================] - 1s 2ms/step
```

In [ ]:

```
print(classification_report(y_test_classes, y_pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.53      0.53      0.53      1000
           1       0.57      0.55      0.56      1000
           2       0.39      0.26      0.31      1000
           3       0.30      0.25      0.27      1000
           4       0.38      0.48      0.43      1000
           5       0.32      0.52      0.40      1000
           6       0.48      0.49      0.49      1000
           7       0.68      0.31      0.43      1000
           8       0.62      0.51      0.56      1000
           9       0.44      0.60      0.51      1000

    accuracy                           0.45     10000
   macro avg       0.47      0.45      0.45     10000
weighted avg       0.47      0.45      0.45     10000
```

In [ ]:

```
# Predict on test data
y_pred = model.predict(X_test)

# Convert predictions from one-hot encoding to class labels
y_pred_classes = np.argmax(y_pred, axis=1)
y_test_classes = np.argmax(y_test, axis=1)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test_classes, y_pred_classes)
precision = precision_score(y_test_classes, y_pred_classes, average='macr
recall = recall_score(y_test_classes, y_pred_classes, average='macro')
f1 = f1_score(y_test_classes, y_pred_classes, average='macro')

# Print results
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
print("F1-Score:", f1)
print("Convergence time", end_time - start_time)
```

```
313/313 [==============================] - 1s 2ms/step
Accuracy: 0.4503
Precision: 0.4709962626586216
Recall: 0.45030000000000003
F1-Score: 0.44787380090393925
Convergence time 70.64562940597534
```

Random Forest Classifier

The CIFAR-10 data has been split into training and test samples using cifar10.load_data() function. The training data is then fit with the random forest classifier and the predictions are done on the test set. The accuracy, precision and recall are calculated for this model as the evaluation metrics.

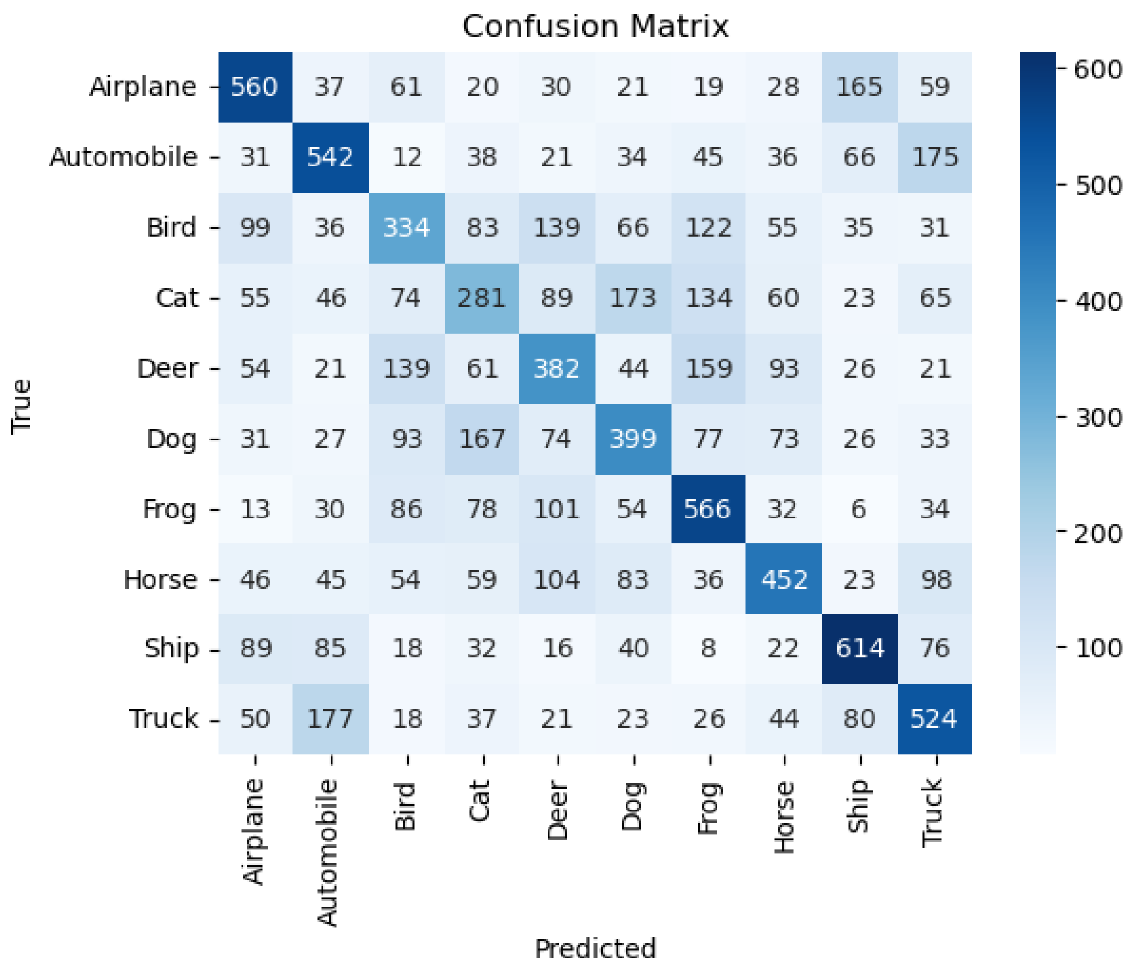
In [ ]:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from keras.datasets import cifar10

# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Flatten the images
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Split the data into train and test sets
X_train, X_test = X_train.astype('float32') / 255.0, X_test.astype('float

# Initialize Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
start_time = time.time()
# Train the model
rf.fit(X_train, y_train)

# Predict on test data
y_pred = rf.predict(X_test)
end_time = time.time()
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Convergence Time:", end_time - start_time)
# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.4654
Convergence Time: 234.11550521850586
Classification Report:
              precision    recall  f1-score   support

           0       0.54      0.56      0.55      1000
           1       0.52      0.54      0.53      1000
           2       0.38      0.33      0.35      1000
           3       0.33      0.28      0.30      1000
           4       0.39      0.38      0.39      1000
```

```
           5       0.43      0.40      0.41      1000
           6       0.47      0.57      0.52      1000
           7       0.51      0.45      0.48      1000
           8       0.58      0.61      0.59      1000
           9       0.47      0.52      0.50      1000

    accuracy                           0.47     10000
   macro avg       0.46      0.47      0.46     10000
weighted avg       0.46      0.47      0.46     10000
```

In [ ]:
```python
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap of the confusion matrix
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d",xticklabels = Classes,
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



SVM With Kernel

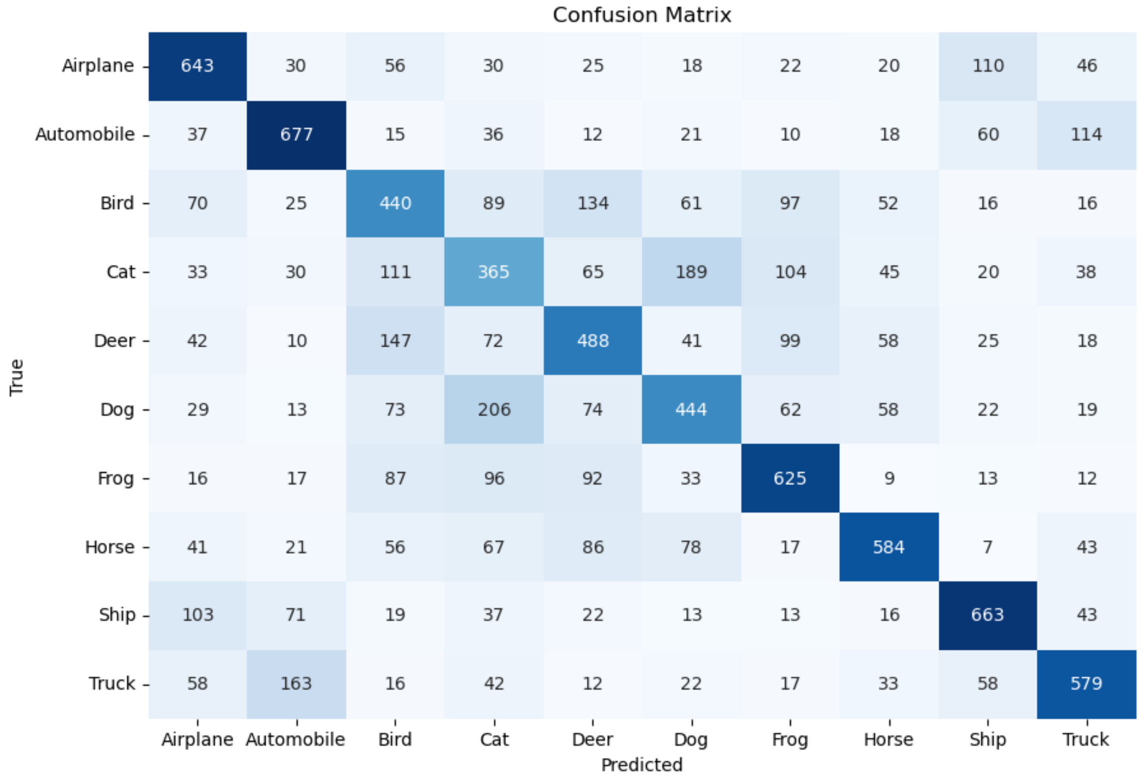
This is a Kernel based SVM which uses the 'rbf' Kernel. First PCA is performed on the dataset for dimensionality reduction. This will improve the performance of SVM on the data. Then a pipeline is created with the PCA component and the SVM to which the data is fed. Once the data is fed into the pipeline the predictions are made

In [ ]:
```python
# Load CIFAR-10 dataset
from keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Flatten the images
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Split the dataset into training and test sets
X_train, X_val_svm, y_train, y_val_svm = train_test_split(X_train, y_trai
```

In [ ]:
```python
# Perform PCA for dimensionality reduction
pca = PCA(n_components=100, random_state=42)

# Create SVM classifier
svm = SVC(kernel='rbf', C=10, random_state=42)

# Create a pipeline with PCA and SVM
pipeline = make_pipeline(StandardScaler(), pca, svm)
start_time = time.time()
# Fit the pipeline to training data
pipeline.fit(X_train, y_train)

# Predict on validation set
y_pred_val_svm = pipeline.predict(X_val_svm)

# Calculate accuracy on validation set
val_accuracy = accuracy_score(y_val_svm, y_pred_val_svm)
print(f'Validation Accuracy: {val_accuracy:.2f}')


# Predict on test set
y_pred_test_svm = pipeline.predict(X_test)
end_time = time.time()
# Calculate accuracy on test set
test_accuracy = accuracy_score(y_test, y_pred_test_svm)
print(f'Test Accuracy: {test_accuracy:.2f}')
print('Convergence time', end_time - start_time)
```

```
Validation Accuracy: 0.55
Test Accuracy: 0.55
Convergence time 318.4526147842407
```

In [ ]:
```python
cm = confusion_matrix(y_test, y_pred_test_svm)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabel
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In [ ]:
```
print(classification_report(y_test, y_pred_test_svm))
```

```
              precision    recall  f1-score   support

           0       0.60      0.64      0.62      1000
           1       0.64      0.68      0.66      1000
           2       0.43      0.44      0.44      1000
           3       0.35      0.36      0.36      1000
           4       0.48      0.49      0.49      1000
           5       0.48      0.44      0.46      1000
           6       0.59      0.62      0.61      1000
           7       0.65      0.58      0.62      1000
           8       0.67      0.66      0.66      1000
           9       0.62      0.58      0.60      1000

    accuracy                           0.55     10000
   macro avg       0.55      0.55      0.55     10000
weighted avg       0.55      0.55      0.55     10000
```

In [ ]:
```
```