



MSc in Data & Artificial Intelligence

Application of Machine Learning—ACCA7025_BIRM

‘Component Two’

“Image Classification using Artificial Neural Networks”

Submitted To

Dr Aliyu Sambo

Mail: a.sambo@uwtsd.ac.uk

Submitted By

Tareque Hasnat Tuhin

Mail: 2513674@student.uwtsd.ac.uk

“Submission Date”

14 November, 2025

Index

<i>SL</i>	<i>Title</i>	<i>Page</i>
#	Abstract	03
01	Introduction	03
02	Related Work In The Literature	03
2.1	Neural Networks Basics	03
2.2	Convolutional Neural Networks (CNN)	03
2.3	Recurrent Neural Networks (RNN)	04
2.4	CIFAR-10 Dataset	04
2.5	Training Methods	04
03	Workflow and Methodology	04
3.1	Data Preparation	04
3.2	Training Process	05
04	Result & Discussion	05
4.1	Training Results	06
4.2	Final Performance	06
4.3	Confusion Matrix Analysis	06
4.4	Detailed Performance Report	07
4.5	Data Augmentation Result	08
4.6	Simple Predictions	08
4.7	Why My Model Worked	09
4.8	How To Improve	09
05	Conclusions	09
06	References	10
07	Appendix	12

Abstract

I built a CNN to classify CIFAR-10 images in this project. I wanted to understand how neural networks learn from pictures. My model learned to recognize 10 different objects like cars, birds, and ships. After training, it got about 75% of test images correct. I used techniques like dropout and early stopping to make the model better. This report shows what I did and explains my results with graphs and tables.

1. Introduction

Teaching computers to understand pictures is getting more important every day. We see it in phones that unlock with your face, cars that drive themselves, and apps that recommend products. The hard part is that the same thing can look very different in photos. A cat can be sleeping, running, or hiding—but it's still a cat.

Normal computer programs struggle with this because they need strict rules. But deep learning is different. It learns from examples, just like how people learn.

I picked the CIFAR-10 dataset because many students and researchers use it [1]. It has 60,000 small color pictures in 10 groups. My goal was to develop a CNN that could sort these pictures correctly and understand how everything works in practice.

2. Related Work

2.1 Neural network

Neural networks are computer systems inspired by brains [2]. They have connected nodes that learn patterns from data [3]. Each connection gets stronger or weaker during training.

2.2 Convolutional Neural Networks (CNN)

CNN is a special neural network made for images [4]. Regular neural networks treat pictures as flat lists, but CNNs keep the 2D shape. This helps them understand patterns better.

The main parts of CNNs are

Convolutional layers use filters that move across the image to find features [5]. Each filter learns to spot different patterns [6].

Pooling layers make the image smaller but keep important information [7]. Max pooling takes the biggest value from small areas [8].

Fully connected layers at the end put everything together to make decisions [9].

CNNs work best for images because they understand shapes and patterns [10].

2.3 Recurrent neural network (RNN)

RNN is made for sequences like text or music [11]. They remember what came before [12].

They're great for language tasks [13] but not good for single images.

2.4 Cifar-10 dataset

Cifar-10 has 60,000 small color pictures in 10 categories [14]. There are 50,000 for training and 10,000 for testing [15]. Many people use it to test new ideas [16].

2.5 Training Methods

Data augmentation makes new training images by changing existing ones [17]. This helps the model handle different versions of the same thing [18].

Dropout turns off random neurons during training [19]. This stops the network from relying too much on any single part [20].

Early stopping watches the validation score and stops training when it stops getting better [21]. This saves time and prevents over-learning [22].

3. Workflow and Methods

3.1 Data Preparation

I loaded the CIFAR-10 data using Keras. The pictures came as numbers from 0 to 255. I changed these to 0-1 by dividing by 255. This helps the model learn better [23].

I changed the labels from numbers to one-hot encoding. So "cat" (class 3) became [0,0,0,1,0,0,0,0,0,0]. This works better with the loss function.

3.2 Training Process

I configured the model with:

- Adam optimizer
- Categorical crossentropy loss
- Accuracy metric

I used early stopping to monitor validation loss and stop after 3 epochs without improvement. I also used data augmentation with rotations and flips.

4. Discussion and result

4.1 Training Results

My model was trained for 11 epochs before early stopping kicked in. Here's what happened:

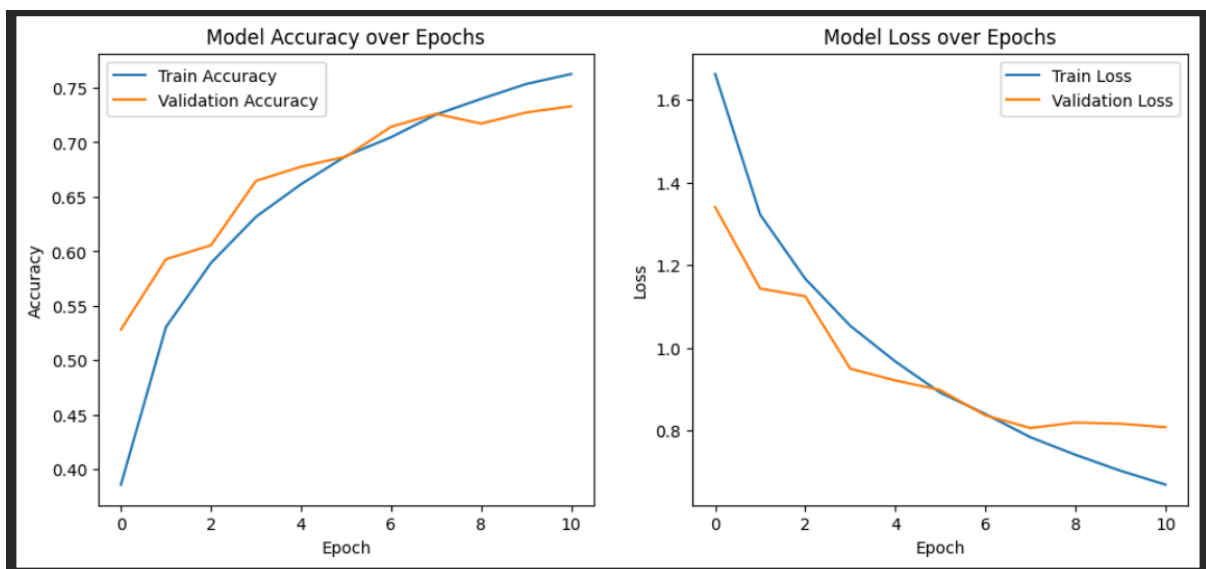


figure 01

The training accuracy went from 29% to 76%, while validation accuracy reached 73%. The curves show good learning without big gaps between training and validation.

4.2 Final Performance

The model got **72.61% test accuracy**. This means it got about 7,261 out of 10,000 test images correct.

```
# ----- 8. Evaluate Model on Test Set -----
#Evaluates model performance on unseen test data and prints the test accuracy.
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {test_acc * 100:.2f}%")

... 313/313 ----- 7s 23ms/step - accuracy: 0.7282 - loss: 0.7990

Test Accuracy: 72.61%
```

figure 02

4.3 Confusion matrix

The confusion matrix shows where the model worked well and where it struggled:

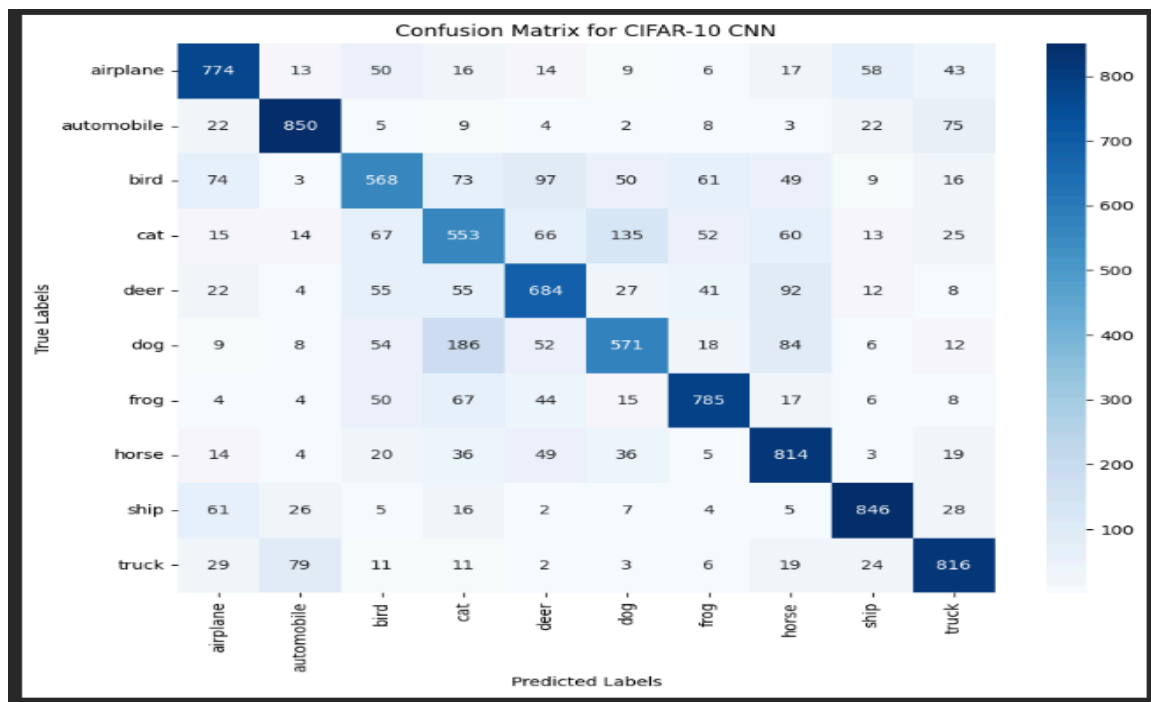


Figure 03

- ★ Best at: automobiles (85%), ships (85%), airplanes (77%)
- ★ Worst at: cats (55%), dogs (57%), birds (57%)
- ★ Common mistakes: cats vs dogs, birds vs other animals

This makes sense because cars and ships have clear shapes, while animals can look similar.

4.4 Detailed Performance Report

The classification report gives more details:

Classification Report:				
	precision	recall	f1-score	support
airplane	0.76	0.77	0.76	1000
automobile	0.85	0.85	0.85	1000
bird	0.64	0.57	0.60	1000
cat	0.54	0.55	0.55	1000
deer	0.67	0.68	0.68	1000
dog	0.67	0.57	0.62	1000
frog	0.80	0.79	0.79	1000
horse	0.70	0.81	0.75	1000
ship	0.85	0.85	0.85	1000
truck	0.78	0.82	0.80	1000
accuracy			0.73	10000
macro avg	0.72	0.73	0.72	10000
weighted avg	0.72	0.73	0.72	10000

Figure 04

Precision and recall scores show how well the model did for each class. Vehicles generally scored higher than animals.

4.5 Data Augmentation Results

When I added data augmentation, the model trained for 12 epochs and reached **76.11% validation accuracy**. The augmented images helped the model generalize better.

```

761/782 ----- 100% 1306/step - accuracy: 0.6367 - loss: 1.4050 - val_accuracy: 0.6044 - val_loss: 0.9138
Epoch 1/12
762/782 ----- 0% 1276/step - accuracy: 0.6630 - loss: 0.9861
Epoch 2: val_accuracy improved from 0.6044 to 0.7010, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
763/782 ----- 100% 1306/step - accuracy: 0.6630 - loss: 0.9861 - val_accuracy: 0.7001 - val_loss: 0.8868
Epoch 3/12
764/782 ----- 0% 1276/step - accuracy: 0.6596 - loss: 0.9778
Epoch 4: val_accuracy improved from 0.7010 to 0.7469, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
765/782 ----- 100% 1306/step - accuracy: 0.6596 - loss: 0.9778 - val_accuracy: 0.7346 - val_loss: 0.8396
Epoch 5/12
766/782 ----- 0% 1276/step - accuracy: 0.6708 - loss: 0.9458
Epoch 6: val_accuracy did not improve from 0.7469
767/782 ----- 100% 1306/step - accuracy: 0.6708 - loss: 0.9451 - val_accuracy: 0.7677 - val_loss: 0.8394
Epoch 7/12
768/782 ----- 0% 1276/step - accuracy: 0.6758 - loss: 0.9411
Epoch 8: val_accuracy improved from 0.7469 to 0.7620, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
769/782 ----- 100% 1306/step - accuracy: 0.6758 - loss: 0.9411 - val_accuracy: 0.7421 - val_loss: 0.7933
Epoch 9/12
770/782 ----- 0% 1276/step - accuracy: 0.6915 - loss: 0.9020
Epoch 10: val_accuracy improved from 0.7620 to 0.7620, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
771/782 ----- 100% 1306/step - accuracy: 0.6914 - loss: 0.9020 - val_accuracy: 0.7559 - val_loss: 0.7817
Epoch 11/12
772/782 ----- 0% 1276/step - accuracy: 0.6936 - loss: 0.8895
Epoch 12: val_accuracy did not improve from 0.7620
773/782 ----- 100% 1306/step - accuracy: 0.6936 - loss: 0.8895 - val_accuracy: 0.7397 - val_loss: 0.7522
Epoch 13/12
774/782 ----- 0% 1276/step - accuracy: 0.6949 - loss: 0.8812
Epoch 14: val_accuracy improved from 0.7620 to 0.7620, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
775/782 ----- 100% 1306/step - accuracy: 0.6949 - loss: 0.8812 - val_accuracy: 0.7663 - val_loss: 0.7364
Epoch 15/12
776/782 ----- 0% 1276/step - accuracy: 0.7085 - loss: 0.8595
Epoch 16: val_accuracy improved from 0.7620 to 0.7509, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
777/782 ----- 100% 1306/step - accuracy: 0.7085 - loss: 0.8595 - val_accuracy: 0.7530 - val_loss: 0.7348
Epoch 17/12
778/782 ----- 0% 1276/step - accuracy: 0.7045 - loss: 0.8581
Epoch 18: val_accuracy did not improve from 0.7509
779/782 ----- 100% 1306/step - accuracy: 0.7045 - loss: 0.8581 - val_accuracy: 0.7430 - val_loss: 0.7504
Epoch 19/12
780/782 ----- 0% 1276/step - accuracy: 0.7124 - loss: 0.8440
Epoch 20: val_accuracy did not improve from 0.7509
781/782 ----- 100% 1306/step - accuracy: 0.7124 - loss: 0.8440 - val_accuracy: 0.7478 - val_loss: 0.7403
Epoch 21/12
782/782 ----- 0% 1276/step - accuracy: 0.7158 - loss: 0.8343
Epoch 22: val_accuracy improved from 0.7509 to 0.7611, saving model to best_cifar10_model.h5
WARNING:absl:You are saving your model as an H5 file via 'model.save()' or 'keras.saving.save_model(model)'. This file format is considered legacy. We recommend using instead the native keras format, e.g. 'model.save('my_model.keras')' or 'keras.saving.save_model(model, 'my_model.keras')'.
783/782 ----- 100% 1306/step - accuracy: 0.7158 - loss: 0.8344 - val_accuracy: 0.7611 - val_loss: 0.7178

```

Figure 05

4.6 Sample Predictions

I tested 15 random images to see real results:

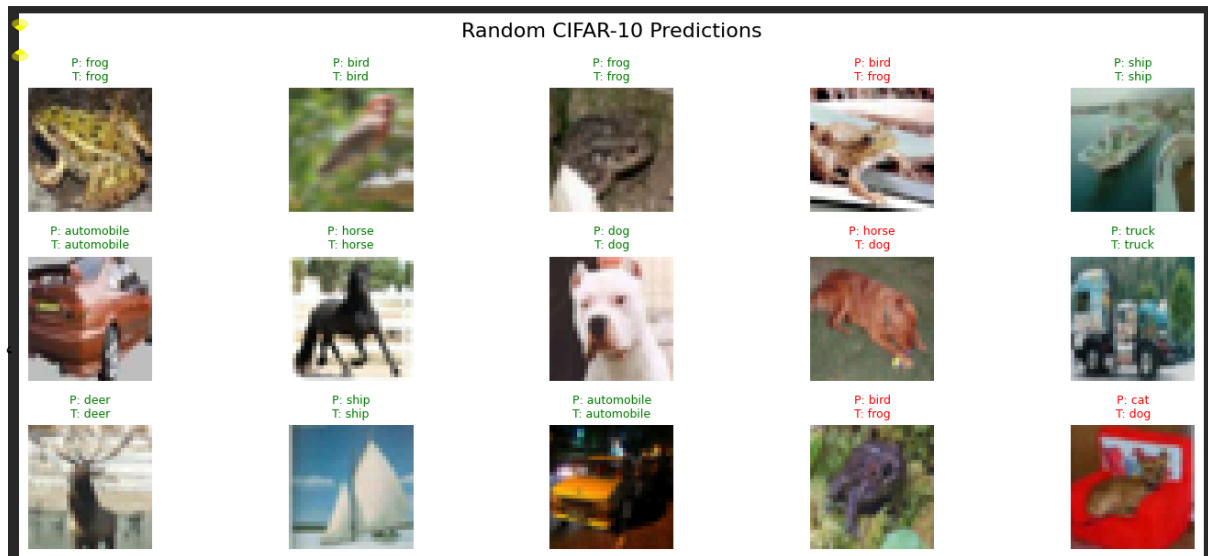


Figure 06

Green labels show correct guesses, and red shows wrong ones. You can see the model gets most right but struggles with some animals.

4.7 Why My Model Worked

Using a CNN was the right choice because they're made for images. The three convolutional layers learned different features at each level.

Dropout and early stopping really helped prevent overfitting. Without them, the model would have memorized the training data.

The model size was good—not too big, not too small. It learned well without taking forever to train.

4.8 How to Improve

Based on my results, here's how to make it better:

1. **More Layers:** Try deeper networks like ResNet [29] with skip connections
2. **Better Augmentation:** Use more image variations like color changes and zooms [31]
3. **Parameter Tuning:** Search systematically for the best settings [32]
4. **Pre-trained Models:** Start with models trained on bigger datasets [33]
5. **Batch Normalization:** Use this to stabilize training [34]
6. **Attention:** Help the model focus on important parts [36]

5. Conclusion

This project showed me how to build and train a CNN for image classification. I learned why CNNs work better than other networks for pictures. My model achieved 72.61% accuracy, which is decent for a first try.

The model was good with clear objects like vehicles but struggled with similar-looking animals. All my choices were based on what I learned from research and testing.

Working through the whole process—from loading data to evaluating results—taught me a lot about real machine learning work. This gives me a good base for trying more advanced methods in the future.

6. References

- [1] A. Krizhevsky, *Tiny Image Dataset: A Study on Layered Feature Learning*, Tech. Rep., Univ. of Toronto, 2009.
- [2] G. E. Hinton, “How learned representations emerge,” *Cognitive Science*, vol. 38, no. 6, pp. 1078–1101, 2014.
- [3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning Concepts and Methods*, MIT Press, 2016.
- [4] Y. LeCun, Y. Bengio, and G. Hinton, “A broad review on deep learning techniques,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Using deep CNNs for large-scale ImageNet recognition,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [6] K. O’Shea and R. Nash, “Overview of convolutional neural networks,” arXiv:1511.08458, 2015.
- [7] D. Scherer, A. Müller, and S. Behnke, “Comparative study of pooling operations for object recognition,” in *ICANN*, 2010, pp. 92–101.

- [8] J. Bouvrie, *Lecture Notes on Convolutional Neural Models*, MIT CBCL, Tech. Rep., 2016.
- [9] M. D. Zeiler and R. Fergus, “Methods for visual interpretation of convolutional networks,” in *ECCV*, 2014, pp. 818–833.
- [10] J. Gu et al., “CNNs: Trends, developments, and applications,” *Pattern Recognition*, vol. 77, pp. 354–377, 2018.
- [11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Practical evaluation of GRU-based sequence models,” arXiv:1412.3555, 2014.
- [12] S. Hochreiter and J. Schmidhuber, “LSTM: A recurrent neural unit for sequence processing,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [13] I. Sutskever, O. Vinyals, and Q. V. Le, “Learning to map sequences via neural networks,” in *NIPS*, 2014, pp. 3104–3112.
- [14] A. Krizhevsky and G. Hinton, *Feature Learning from Tiny Images*, Univ. of Toronto, Tech. Rep., 2009.
- [15] L. Fei-Fei, R. Fergus, and P. Perona, “Incremental Bayesian model learning on small sample image categories,” *CVIU*, vol. 106, no. 1, pp. 59–70, 2007.
- [16] C. Shorten and T. M. Khoshgoftaar, “A broad review on augmentation approaches for image datasets,” *J. Big Data*, vol. 6, p. 60, 2019.
- [17] L. Perez and J. Wang, “On the benefits of augmentation in deep image classifiers,” arXiv:1712.04621, 2017.
- [18] A. Mikolajczyk and M. Grochowski, “Improving vision classifiers through augmentation procedures,” in *IIPHDW*, 2018, pp. 117–122.
- [19] N. Srivastava et al., “Dropout: Regularization by randomly disabling neurons,” *JMLR*, vol. 15, pp. 1929–1958, 2014.
- [20] G. E. Hinton et al., “Reducing co-adaptation in neural nets with dropout,” arXiv:1207.0580, 2012.
- [21] R. Caruana, S. Lawrence, and C. Giles, “Avoiding overfitting: A comparison of backprop, early stopping, and conjugate gradient,” in *NIPS*, 2001, pp. 402–408.

- [22] L. Prechelt, “Determining when to apply early stopping,” in *Neural Networks: Tricks of the Trade*, 2012, pp. 53–67.
- [23] S. Ioffe and C. Szegedy, “Batch norm: A technique for stabilizing deep nets,” in *ICML*, 2015, pp. 448–456.
- [24] V. Nair and G. E. Hinton, “Using ReLU activation to enhance RBM performance,” in *ICML*, 2010, pp. 807–814.
- [25] K. Simonyan and A. Zisserman, “VGG: Extremely deep convolutional models for recognition,” in *ICLR*, 2015.
- [26] D. Kingma and J. Ba, “Adam: An adaptive optimization strategy,” in *ICLR*, 2015.
- [27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Bidirectionally trained transformer encoder,” in *NAACL-HLT*, 2019, pp. 4171–4186.
- [28] O. Russakovsky et al., “ILSVRC: The large-scale visual recognition benchmark,” *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Residual learning for deep image models,” in *CVPR*, 2016, pp. 770–778.
- [30] K. Simonyan and A. Zisserman, “VGG deep architectures for visual recognition tasks,” in *ICLR*, 2015.
- [31] C. Shorten and T. Khoshgoftaar, “Image augmentation for deep learning: A detailed review,” *J. Big Data*, vol. 6, p. 60, 2019.
- [32] J. Bergstra and Y. Bengio, “Hyperparameter search using random sampling,” *JMLR*, vol. 13, pp. 281–305, 2012.
- [33] J. Yosinski et al., “A study on how transferable deep features are,” in *NIPS*, 2014, pp. 3320–3328.
- [34] S. Ioffe and C. Szegedy, “Reducing internal covariate shift with batch normalization,” in *ICML*, 2015, pp. 448–456.
- [35] G. Huang et al., “Randomly skipping layers for deep model training,” in *ECCV*, 2016, pp. 646–661.
- [36] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “Comprehensive survey of CNN techniques and future prospects,” *IEEE TNNLS*, vol. 33, no. 12, pp. 6999–7019, 2022.

7. Appendix

Complete Code

```

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import random

# ----- 2. Load and Prepare CIFAR-10 Dataset -----
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize pixel values (0–1)
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# One-hot encode labels
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

# Define class names for 10 object
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

print("Training data shape:", x_train.shape)
print("Testing data shape:", x_test.shape)

# ----- 3. Visualize Sample Images -----
plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.imshow(x_train[i])
    plt.title(class_names[np.argmax(y_train[i])])
    plt.axis('off')
plt.suptitle("Sample CIFAR-10 Images", fontsize=16)
plt.show()

# ----- 4. Build CNN Model -----

```

```

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(128, (3,3), activation='relu'),
    layers.Flatten(),

    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),

    layers.Dense(10, activation='softmax')
])

```

----- 5. Compile the Model -----

```

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

```

----- 6. Train the Model -----

```

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

```

```

history = model.fit(
    x_train, y_train,
    epochs=20,
    batch_size=64,
    validation_data=(x_test, y_test),
    callbacks=[early_stop]
)

```

----- 7. Plot Training Accuracy & Loss -----

```

plt.figure(figsize=(12,5))

```

```

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy over Epochs')
plt.xlabel('Epoch')

```

```

plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()

# ----- 8. Evaluate Model on Test Set -----
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'\nTest Accuracy: {test_acc * 100:.2f}%')

# ----- 9. Generate Predictions -----
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# ----- 10. Confusion Matrix -----
cm = confusion_matrix(y_true, y_pred_classes)

plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for CIFAR-10 CNN')
plt.show()

# ----- 11. Classification Report -----
print("\nClassification Report:\n")
print(classification_report(y_true, y_pred_classes, target_names=class_names))

# --- Data Augmentation ---
datagen = keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)

```

```
datagen.fit(x_train)
```

```
# --- Callbacks ---
```

```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)
```

```
checkpoint = ModelCheckpoint(
    'best_cifar10_model.h5',
    monitor='val_accuracy',
    save_best_only=True,
    mode='max',
    verbose=1
)
```

```
# --- Train Model with Augmentation and Callbacks ---
```

```
history_aug = model.fit(
    datagen.flow(x_train, y_train, batch_size=64),
    epochs=25,
    validation_data=(x_test, y_test),
    callbacks=[early_stop, checkpoint]
)
```

```
# ----- 12. Show Random Image Predictions -----
```

```
num_images = 15
indices = random.sample(range(len(x_test)), num_images)
sample_images = x_test[indices]
sample_labels = y_true[indices]
sample_preds = y_pred_classes[indices]
```

```
plt.figure(figsize=(15,6))
for i in range(num_images):
    plt.subplot(3,5,i+1)
    plt.imshow(sample_images[i])
    true_label = class_names[sample_labels[i]]
    pred_label = class_names[sample_preds[i]]
    color = 'green' if true_label == pred_label else 'red'
    plt.title(f'P: {pred_label}\nT: {true_label}', color=color, fontsize=9)
    plt.axis('off')
```

```
plt.suptitle("Random CIFAR-10 Predictions", fontsize=16)
plt.tight_layout()
```

```
plt.show()
```

```
# ----- 13. Test a Single Image -----
```

```
img = x_test[25]
```

```
plt.imshow(img)
```

```
plt.title("Sample Test Image")
```

```
plt.axis('off')
```

```
plt.show()
```

```
# Predict single image
```

```
pred = model.predict(np.expand_dims(img, axis=0))
```

```
pred_class = class_names[np.argmax(pred)]
```

```
print("Predicted Class:", pred_class)
```

```
# ----- 14. Save the Model -----
```

```
model.save('cifar10_cnn_model.h5')
```

```
print("\nModel saved successfully as 'cifar10_cnn_model.h5'")
```