Kingdom of Saudi Arabia
Ministry of Education
Saudi Digital Academy

PROJECT

Project title:	Deep learning – image classification with cnn
Organization:	Saudi Digital Academy
Objectives:	Project number 2
Undertaken By:	Essa alhassan Abdulaziz Alshehri Razan Khalil Mlk alshaikh
Date Started:	4\3\2025
Date Completed:	7\3\2025
Technologies Used:	Colap , vs
Operating System:	Windows ,apple silicon

Introduction

• **Introduction (cnn) :** In this project, we will learn how to apply a convolutional neural network (CNN) to a dataset. It is a type of artificial neural network, mainly used in image processing and pattern recognition. CNNs are very effective in tasks such as image classification, object detection, and video analysis. It may contain main components: Convolutional Layers: Function: Used to extract features from images, such as edges and shapes. Step two Pooling Layers: Function: Used to reduce the spatial dimensions of the data, which reduces the number of features and reduces computational complexity. The most common type of pooling is Max Pooling. It was used in this project. Step three Activation Layers: Function: Used to introduce nonlinearity into the model. The ReLU (Rectified Linear Unit) function is one of the most commonly used functions. Step four Fully Connected Layers: Function: At the end of the network, these layers are used to connect the extracted features to the final results (such as image classification). last step Dropout: Its function: A technique used to reduce overfitting by setting a percentage of units to zero during training.

After using the CNN The transfer learning phase begins Transfer learning is a technique in which the knowledge gained from a previously trained model (such as a

CNN model) on a specific dataset is used to speed up the training process or improve the performance of a new model on a different dataset. Then we compare these two models to see if the best model we used is the CNN or the transfer

• Introduction (Transfor) :

Transfer Learning is a machine learning technique that uses the knowledge gained from a model that has been previously trained on a specific task, to facilitate the training of a new model on a different, but related task.
In other words, instead of training a model from scratch, we use a pre-trained model (such as a CNN trained on a large dataset such as ImageNet) and modify it for a new task.
Basic steps:
1. **Select a pre-trained model**: We start with a model that has strong weights and learning foundations (such as VGG or ResNet).
2. **Modify the model**: We remove the last layers (that are relevant to the original task) and add new layers that are relevant to the new task.
3. **Retrain**: We train the modified model on a new dataset, where we can use a small dataset.
Benefits:
- **Time saving**: It reduces the time and resources required to train the model.
- **Performance improvement**: It helps improve the accuracy of the model, especially if the new dataset is small.
In short, Transfer Learning is an effective way to leverage pre-trained models to achieve better performance on new tasks.

• Problem Statement

Data complexity: Images and videos contain huge amounts of data. Traditional models cannot handle this amount of data effectively.

Data bias: Some models may be very sensitive to biases in the data.

Pattern identification: Traditional models find it difficult to identify complex patterns in data.

Multi-scale: Images may contain features of different scales.

● Objectives

● Improving classification accuracy: The study aims to use CNN to achieve higher accuracy in image classification compared to traditional methods, which helps improve the results of various applications such as face recognition and image classification.

● accelerating the learning process: By exploiting the deep structure of neural networks, we seek to reduce the time spent on training and improve the efficiency of the model.

● Developing flexible models: We aim to create models that can adapt to a variety of data, including images of different quality and dimensions.

● achieving deep feature learning: The goal is to improve the model's ability to extract essential features from data automatically, reducing the need for manual processing.

● Reducing bias in data: We seek to develop models that can reduce the impact of biases present in training data, ensuring more accurate and reliable results.

● improving generalization ability: We aim to enhance the model's ability to generalize to new, unfamiliar data, making the results more resilient in practical applications.

Providing practical solutions to real-world problems: By using CNN, we seek to provide solutions to practical challenges in areas such as healthcare, security, and e-commerce, which contributes to improving performance and efficiency.

● Methods

There are some methods that help to achieve the project goals effectively and improve the model's performance in processing data using convolutional neural networks, including:

Convolutional Neural Networks (CNN)

- Convolutional Layers: used to extract features from images by applying filters to the input data, allowing local patterns to be captured.

- Pooling Layers: used to reduce the dimensions of the data resulting from the convolutional layers, which helps reduce the number of parameters and increases the efficiency of the model.
- Activation Layers: such as the ReLU function, used to introduce non-linearity into the model, which helps the network learn more complex features.
- Fully Connected Layers: used ultimately to classify the extracted features, as they connect all cells in the previous layer to cells in the next layer.

2. Transfer Learning

- Transferring knowledge from a pre-trained model: A model that has been previously trained on a large dataset (such as ImageNet) is used as a base, and the latter layers are modified to suit the new task.
- Freezing and tuning: The first layers can be "frozen" (untrained) to retain the extracted features, while the upper layers are tweaked to fit new data.
- Fine-tuning: After training the model on new data, the entire model or some layers may be retrained to improve performance.

3. Performance improvement techniques

- Data Augmentation: Techniques such as rotating, cropping, or changing the brightness of images to increase the diversity of the data used in training, which helps improve the generalization of the model.
- Preprocessing: Techniques such as converting images to grayscale, resizing, or normalizing them before feeding them into the model.
- Multi-task Learning: Training the model on multiple tasks simultaneously to improve overall performance and learning efficiency.

4. Tools and Applications

Frameworks used: such as TensorFlow and PyTorch, which provide powerful tools for developing and training CNN models and transfer learning.

Ready libraries: such as Keras, which facilitate building neural networks and applying transfer learning techniques.

• Proposed Solution

- Automatic feature optimization: CNNs automatically extract features from data, reducing the need for manual feature selection.
- Flexibility in learning: Thanks to their deep structure, CNNs can learn from large amounts of data and improve their performance over time.
- Good generalization: CNNs improve the model's ability to generalize to new data that was not used in training, making them effective in practical applications.
- Noise handling: CNNs are able to recognize patterns even in the presence of noise in the data, making them ideal for real-world applications.

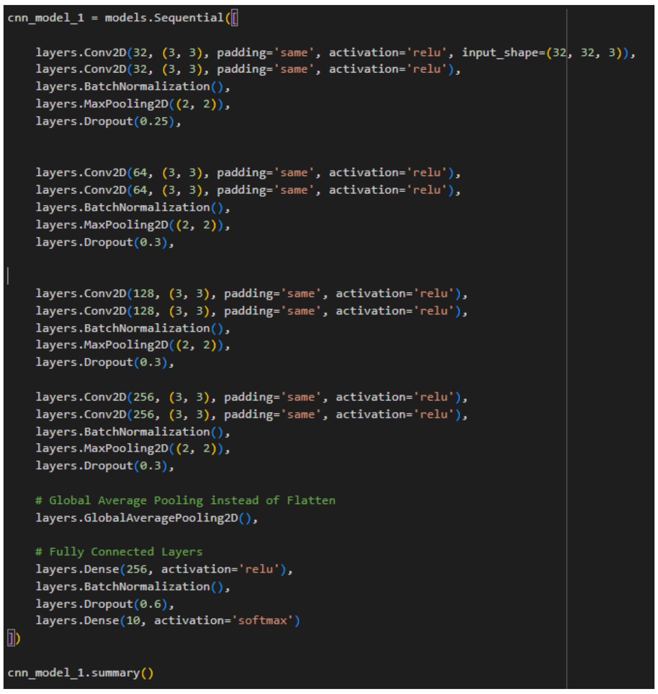
CONTENTS

LIST OF FIGURES

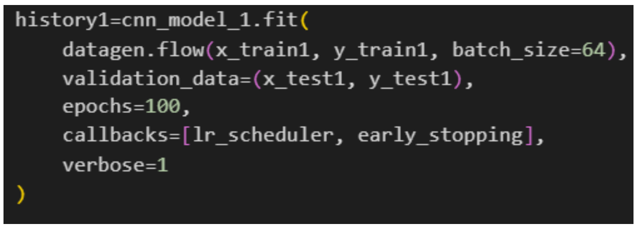
LIST OF TABLES

IMPLEMENTATION AND OUTPUT

CNN (model_1)

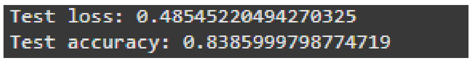


```
cnn_model_1 = models.Sequential([

    layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32, 32, 3)),
    layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.25),


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


    layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
    layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    # Global Average Pooling instead of Flatten
    layers.GlobalAveragePooling2D(),

    # Fully Connected Layers
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.6),
    layers.Dense(10, activation='softmax')
])

cnn_model_1.summary()
```



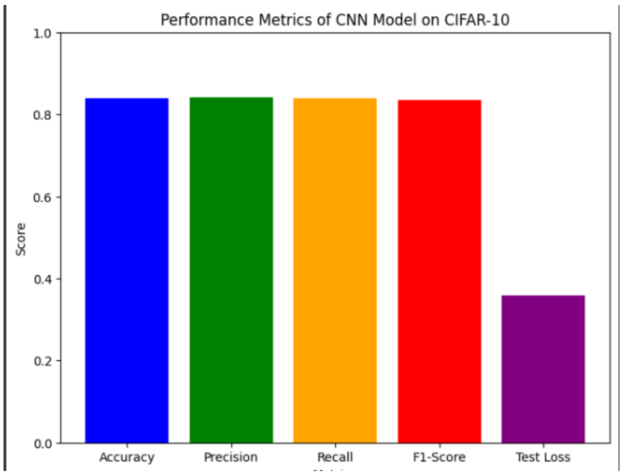
```
history1=cnn_model_1.fit(
    datagen.flow(x_train1, y_train1, batch_size=64),
    validation_data=(x_test1, y_test1),
    epochs=100,
    callbacks=[lr_scheduler, early_stopping],
    verbose=1
)
```

➔ Output 1.1

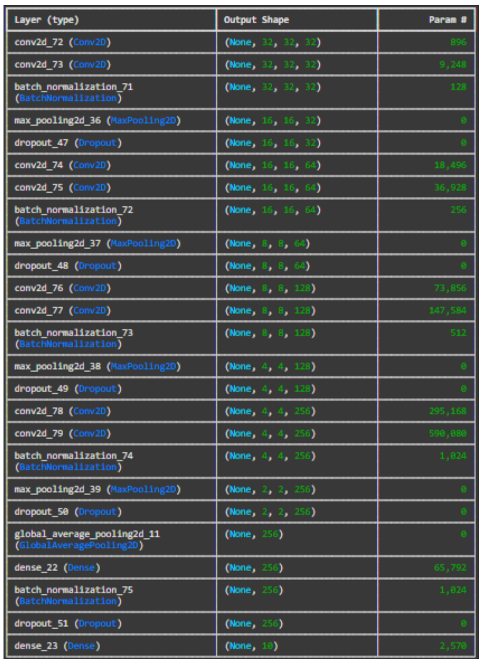


```
Test loss: 0.48545220494270325
Test accuracy: 0.8385999798774719
```

➔ figure 1.1

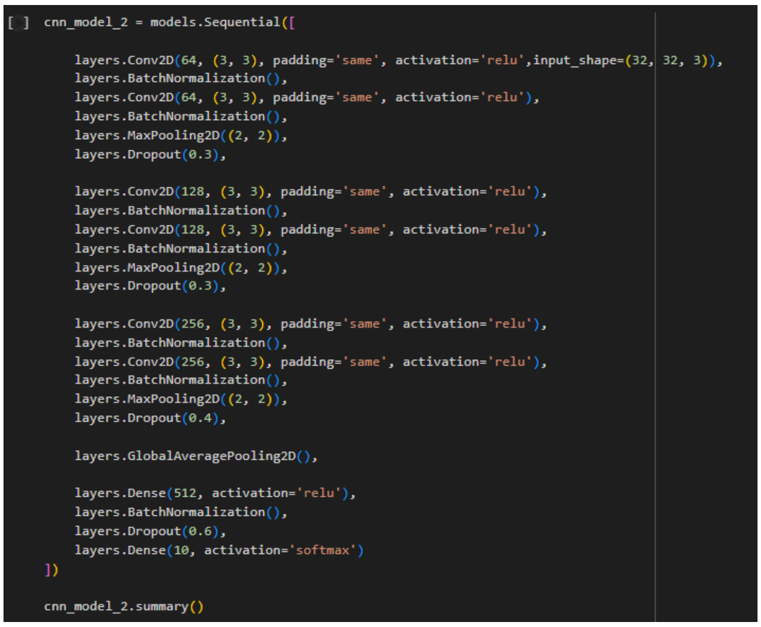


➔ table 1.1

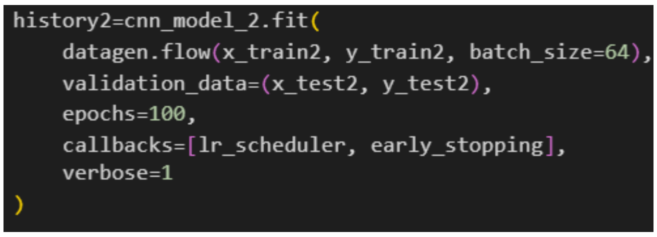


Layer (type)	Output Shape	Param #
conv2d_72 (Conv2D)	(None, 32, 32, 32)	896
conv2d_73 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_71 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d_36 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_47 (Dropout)	(None, 16, 16, 32)	0
conv2d_74 (Conv2D)	(None, 16, 16, 64)	18,496
conv2d_75 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_72 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_37 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_48 (Dropout)	(None, 8, 8, 64)	0
conv2d_76 (Conv2D)	(None, 8, 8, 128)	73,856
conv2d_77 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_73 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_38 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_49 (Dropout)	(None, 4, 4, 128)	0
conv2d_78 (Conv2D)	(None, 4, 4, 256)	295,168
conv2d_79 (Conv2D)	(None, 4, 4, 256)	590,080
batch_normalization_74 (BatchNormalization)	(None, 4, 4, 256)	1,024
max_pooling2d_39 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_50 (Dropout)	(None, 2, 2, 256)	0
global_average_pooling2d_11 (GlobalAveragePooling2D)	(None, 256)	0
dense_22 (Dense)	(None, 256)	65,792
batch_normalization_75 (BatchNormalization)	(None, 256)	1,024
dropout_51 (Dropout)	(None, 256)	0
dense_23 (Dense)	(None, 10)	2,570

CNN (model_2)

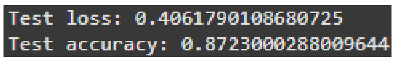


```
[ ]  cnn_model_2 = models.Sequential([

         layers.Conv2D(64, (3, 3), padding='same', activation='relu',input_shape=(32, 32, 3)),
         layers.BatchNormalization(),
         layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
         layers.BatchNormalization(),
         layers.MaxPooling2D((2, 2)),
         layers.Dropout(0.3),

         layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
         layers.BatchNormalization(),
         layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
         layers.BatchNormalization(),
         layers.MaxPooling2D((2, 2)),
         layers.Dropout(0.3),

         layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
         layers.BatchNormalization(),
         layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
         layers.BatchNormalization(),
         layers.MaxPooling2D((2, 2)),
         layers.Dropout(0.4),

         layers.GlobalAveragePooling2D(),

         layers.Dense(512, activation='relu'),
         layers.BatchNormalization(),
         layers.Dropout(0.6),
         layers.Dense(10, activation='softmax')
     ])

     cnn_model_2.summary()
```



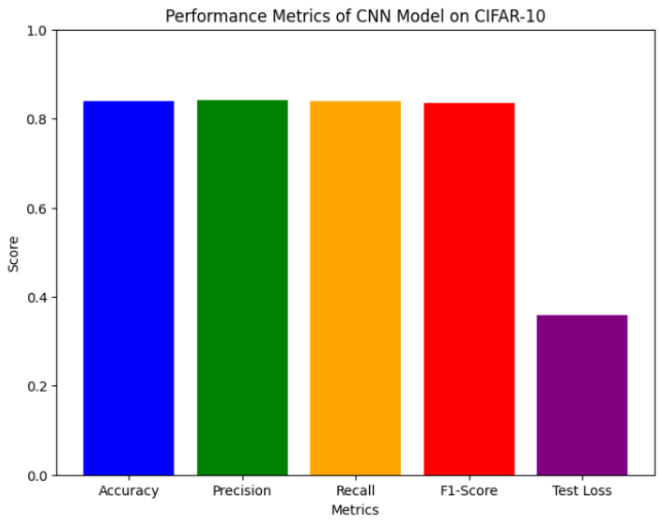
```
history2=cnn_model_2.fit(
    datagen.flow(x_train2, y_train2, batch_size=64),
    validation_data=(x_test2, y_test2),
    epochs=100,
    callbacks=[lr_scheduler, early_stopping],
    verbose=1
)
```

➔ Output 1.2

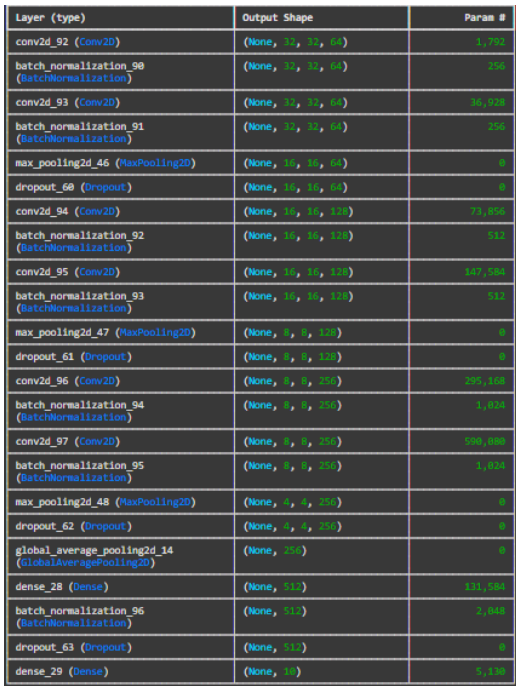


```
Test loss: 0.4061790108680725
Test accuracy: 0.8723000288009644
```

➔ figure 1.2

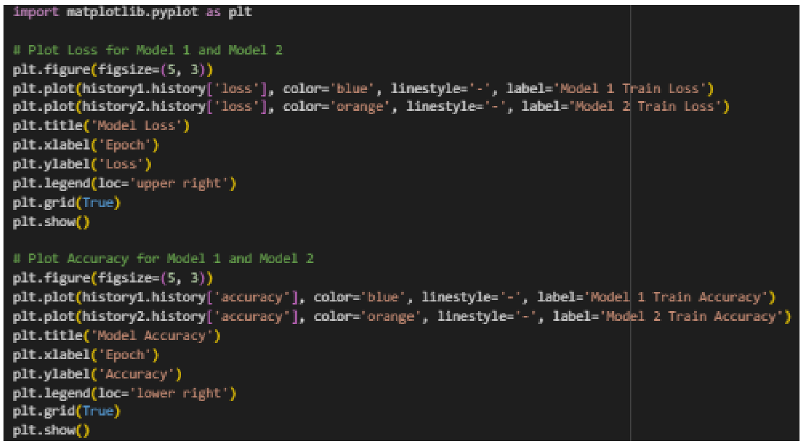


➔ table 1.2



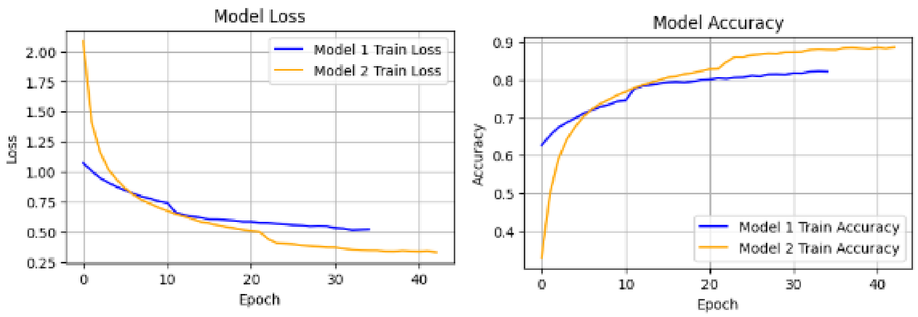
Layer (type)	Output Shape	Param #
conv2d_92 (Conv2D)	(None, 32, 32, 64)	1,792
batch_normalization_90 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_93 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_91 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_46 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_60 (Dropout)	(None, 16, 16, 64)	0
conv2d_94 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_92 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_95 (Conv2D)	(None, 16, 16, 128)	147,584
batch_normalization_93 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_47 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_61 (Dropout)	(None, 8, 8, 128)	0
conv2d_96 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_94 (BatchNormalization)	(None, 8, 8, 256)	1,024
conv2d_97 (Conv2D)	(None, 8, 8, 256)	590,080
batch_normalization_95 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_48 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_62 (Dropout)	(None, 4, 4, 256)	0
global_average_pooling2d_14 (GlobalAveragePooling2D)	(None, 256)	0
dense_28 (Dense)	(None, 512)	131,584
batch_normalization_96 (BatchNormalization)	(None, 512)	2,048
dropout_63 (Dropout)	(None, 512)	0
dense_29 (Dense)	(None, 10)	5,130

COMPARE two model(CNN)

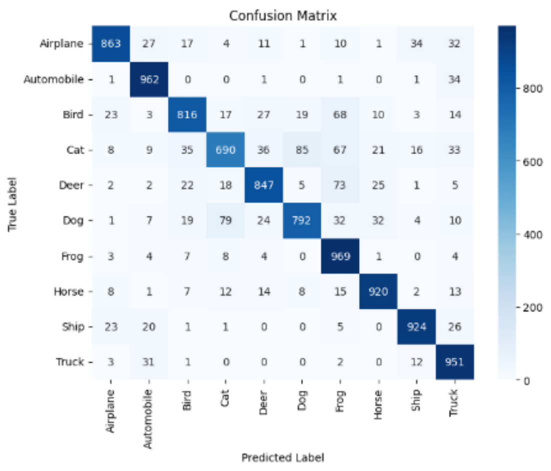


```python
import matplotlib.pyplot as plt

# Plot Loss for Model 1 and Model 2
plt.figure(figsize=(5, 3))
plt.plot(history1.history['loss'], color='blue', linestyle='-', label='Model 1 Train Loss')
plt.plot(history2.history['loss'], color='orange', linestyle='-', label='Model 2 Train Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()

# Plot Accuracy for Model 1 and Model 2
plt.figure(figsize=(5, 3))
plt.plot(history1.history['accuracy'], color='blue', linestyle='-', label='Model 1 Train Accuracy')
plt.plot(history2.history['accuracy'], color='orange', linestyle='-', label='Model 2 Train Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

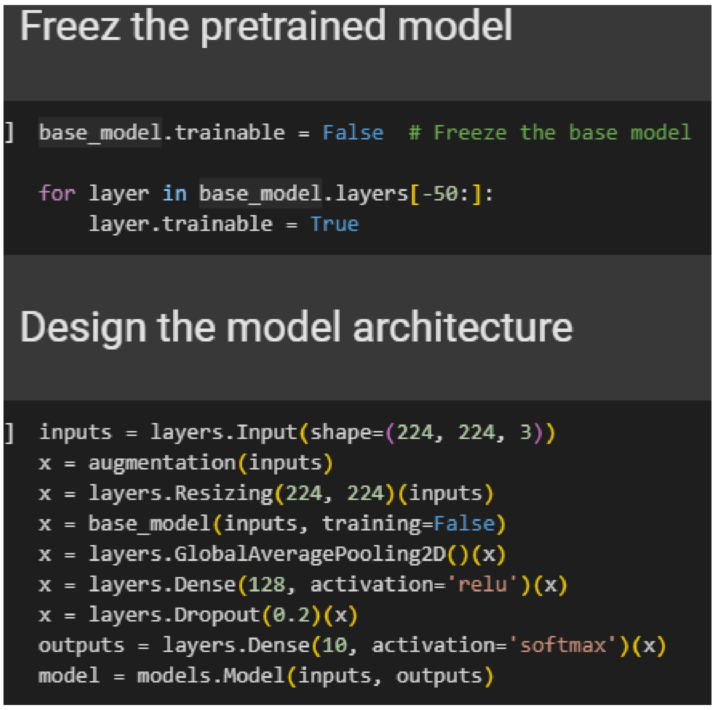
➔ figure 1.3



➔ figure 1.4



TRANSFER (model_MobileNest)

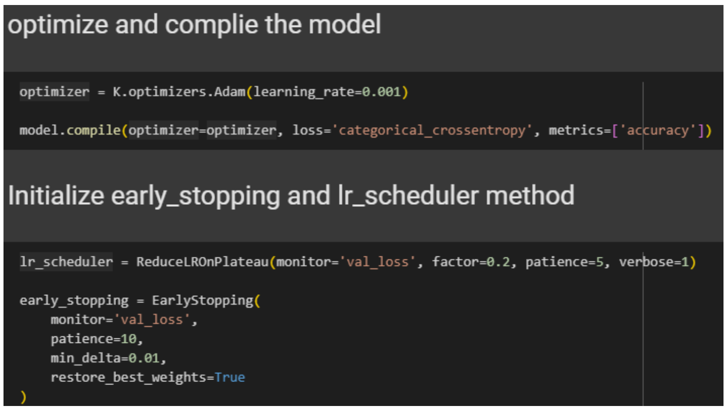


Freez the pretrained model

```
]  base_model.trainable = False  # Freeze the base model

   for layer in base_model.layers[-50:]:
       layer.trainable = True
```

Design the model architecture

```
]  inputs = layers.Input(shape=(224, 224, 3))
   x = augmentation(inputs)
   x = layers.Resizing(224, 224)(inputs)
   x = base_model(inputs, training=False)
   x = layers.GlobalAveragePooling2D()(x)
   x = layers.Dense(128, activation='relu')(x)
   x = layers.Dropout(0.2)(x)
   outputs = layers.Dense(10, activation='softmax')(x)
   model = models.Model(inputs, outputs)
```



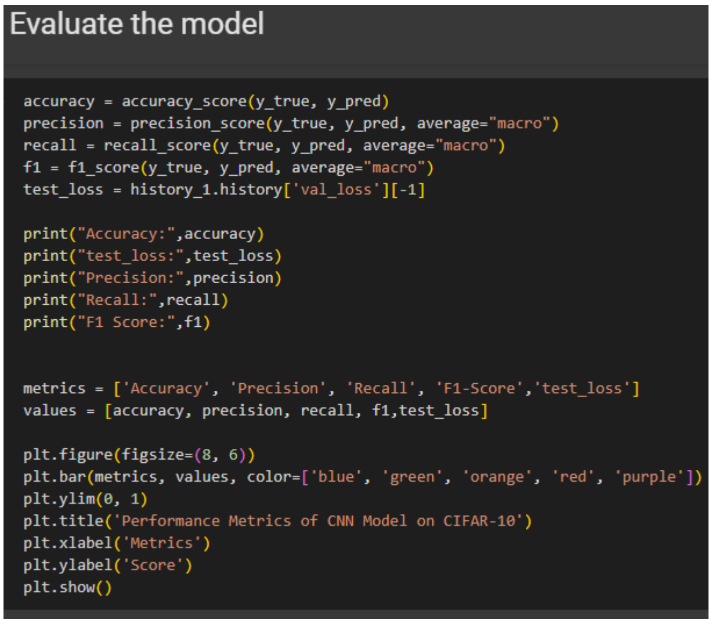
optimize and complie the model

```
optimizer = K.optimizers.Adam(learning_rate=0.001)

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

Initialize early_stopping and lr_scheduler method

```
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    min_delta=0.01,
    restore_best_weights=True
)
```

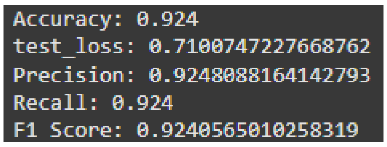
```
history_1 = model.fit(train_dataset, validation_data=test_dataset,
epochs=100, callbacks=[early_stopping, lr_scheduler])
```



Evaluate the model

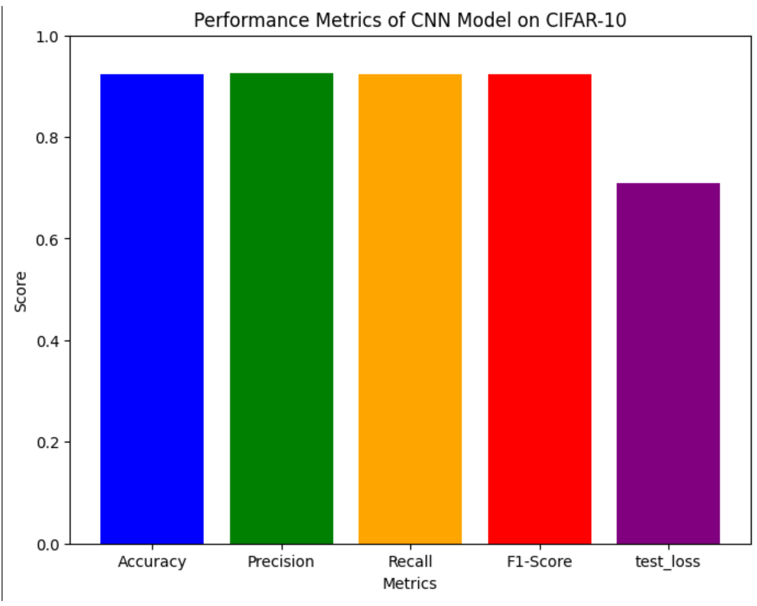
```
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average="macro")
recall = recall_score(y_true, y_pred, average="macro")
f1 = f1_score(y_true, y_pred, average="macro")
test_loss = history_1.history['val_loss'][-1]

print("Accuracy:",accuracy)
print("test_loss:",test_loss)
print("Precision:",precision)
print("Recall:",recall)
print("F1 Score:",f1)


metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score','test_loss']
values = [accuracy, precision, recall, f1,test_loss]

plt.figure(figsize=(8, 6))
plt.bar(metrics, values, color=['blue', 'green', 'orange', 'red', 'purple'])
plt.ylim(0, 1)
plt.title('Performance Metrics of CNN Model on CIFAR-10')
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.show()
```

➔ Output 2.1

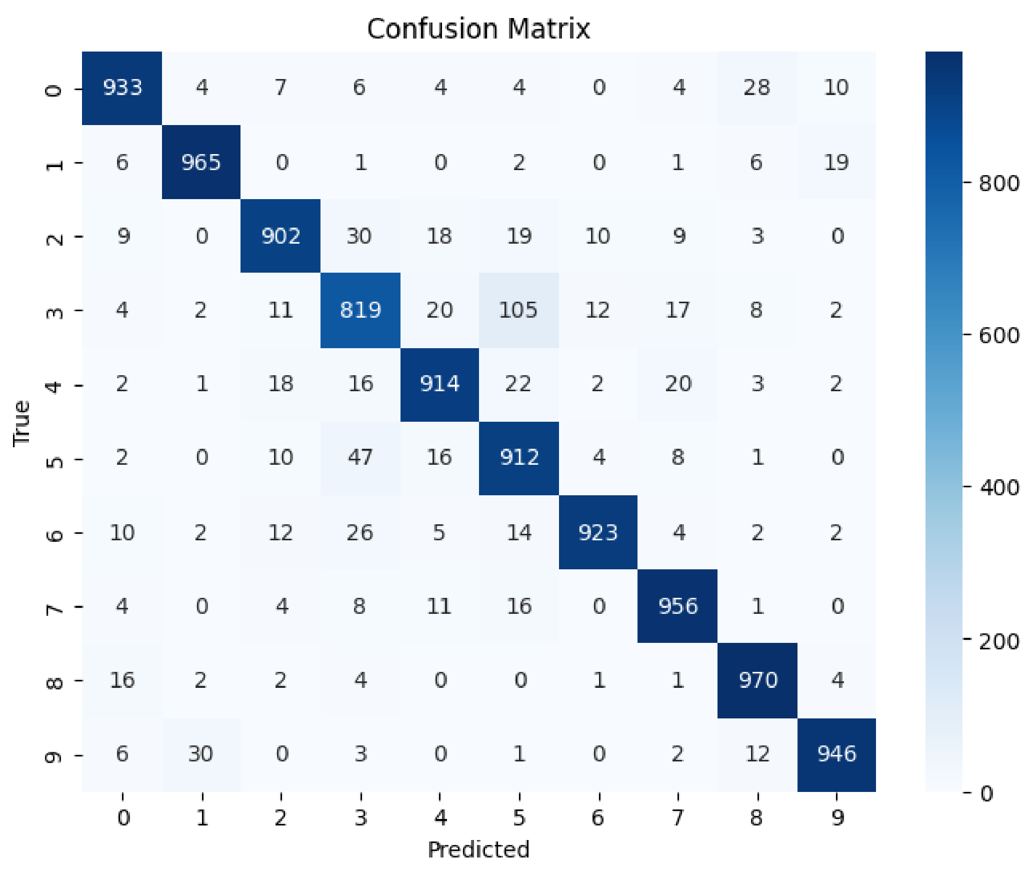


```
Accuracy: 0.924
test_loss: 0.7100747227668762
Precision: 0.9248088164142793
Recall: 0.924
F1 Score: 0.9240565010258319
```

➔ figure 2.1

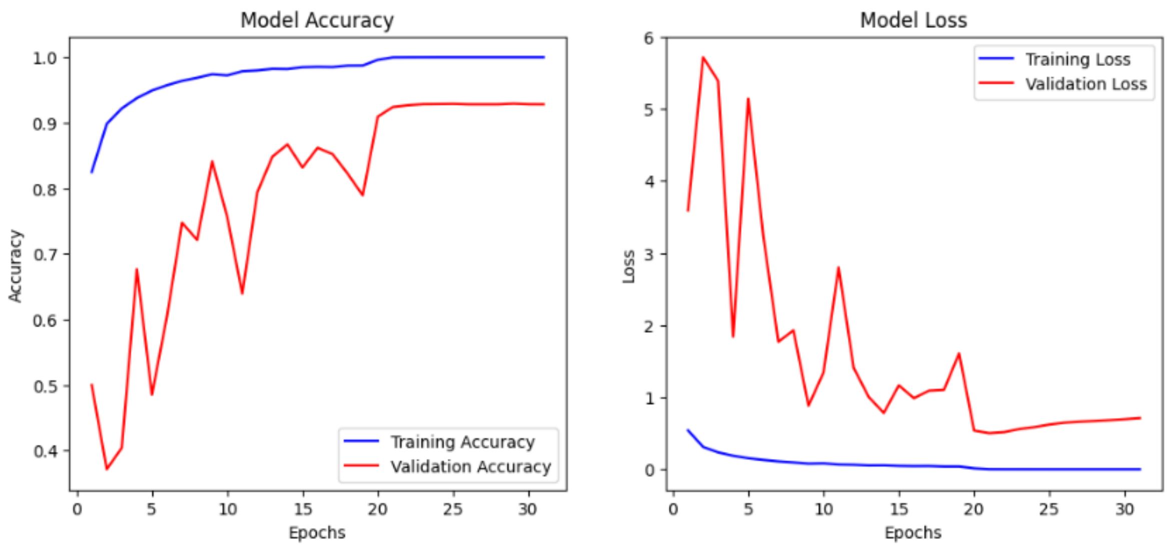


➔ figure 2.2



COMPARE (CNN , TRANSFER)

➔ figure 3.1



CONCLUSION AND REFERENCES

CONCLUSION

Convolutional neural networks (CNNs) and transfer learning are fundamental tools that have revolutionized the field of deep learning, especially in applications related to computer vision and image processing. With their ability to extract features automatically and reduce the need for large datasets, these techniques provide effective solutions to the challenges associated with model training. Using CNNs, we can achieve high accuracy in various tasks, while transfer learning helps speed up the process and reduce costs. Combining these two approaches allows us to leverage the knowledge gained from pre-trained models, facilitating the scaling of applications in multiple fields such as medicine, security, and commerce.

REFERENCES

https://www.kaggle.com/code/gpreda/cats-or-dogs-using-cnn-with-transfer-learning

https://www.kaggle.com/code/dansbecker/transfer-learning

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

https://www.kaggle.com/code/samanyuk/fire-94-accuracy