Neural network Models for object recognition on CIFAR-10 Dataset

By: Amaka Ndudirim
Student ID #31522

INTRODUCTION

- **Focus**: Deep learning:
  - Convolutional Neural Networks(CNNs)
  - Data Augmentation
  - Transfer Learning and Fine-tuning of pre-trained models
- **Objective**: To develop and evaluate an object recognition model on CIFAR-10 using deep learning.
- **Project Aim**: Design, train, and evaluate deep learning models to understand performance trade-offs between a custom CNN and a transfer-learned MobileNetV2 model on CIFAR-10.

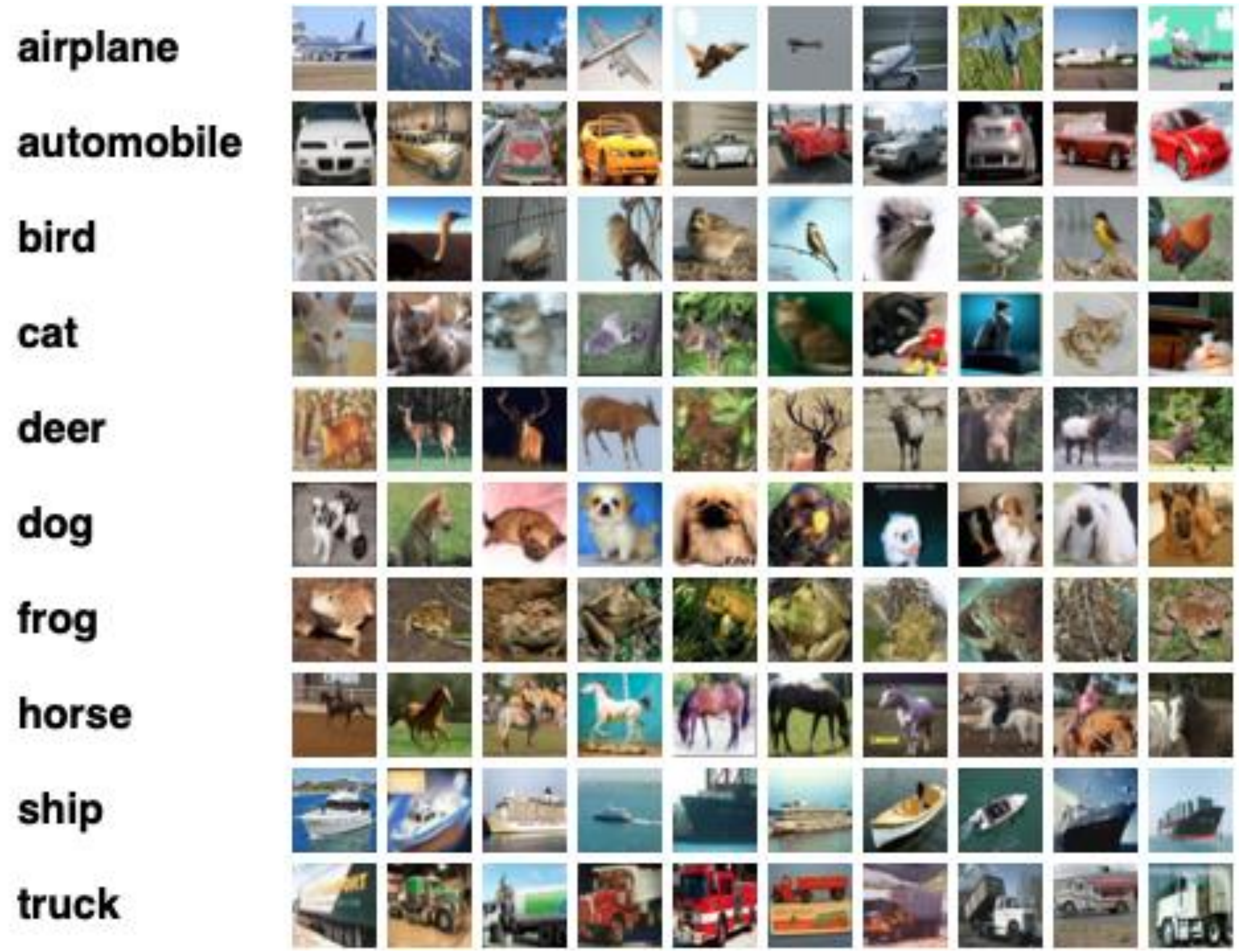


Figure 1. Example images from the CIFAR-10 dataset showing ten object classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).
Source: Krizhevsky, A., Nair, V. & Hinton, G. (2009). Learning Multiple Layers of Features from Tiny Images. University of Toronto, Dataset available at https://www.cs.toronto.edu/~kriz/cifar.html.

DATA PREPARATION

Dataset: CIFAR-10 with 60,000 color images(32x32, 10 classes)

Splitting:

Training: 50,000 images split into

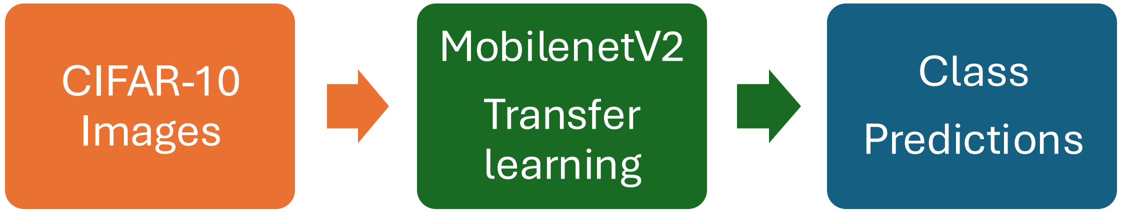
Train 45,000

Validation 5,000

Test: 10,000 images

Class Distribution: train & test sets uniformly distributed across 10 labels (see bar chart)

MODEL ARCHITECHTURE



Custom CNN (from scratch)

- A simple sequential CNN built using Conv2D → ReLU → MaxPooling layers.
- Ends with dense layers and a Softmax output for 10 classes (Krizhevsky, Nair & Hinton, 2009).
- Roughly **1–2 million parameters.**
- Trained using **Adam, cross-entropy loss**, and Xavier weight initialization.

MobileNetV2 (Transfer Learning)

- Used **pre-trained ImageNet weights** as a fixed feature extractor (Howard et al., 2017) .
- Added a new classification head (Dense layer with 10 outputs).
- Around **3.5 million parameters**, most in the base network.
- Initially trained only the top classifier, then **fine-tuned the top layers** of the base with a smaller learning rate (LeCun, Bengio & Hinton, 2015).

Training Pipeline

- Augmentation applied to both datasets.
- **CNN**: Learned features from scratch — started slow, required more epochs.
- **MobileNetV2**: Leveraged pre-learned filters, converged faster and reached higher accuracy with fewer epochs (Howard et al., 2017) .

Automatically Learned vs Manually Selected Hyperparameters

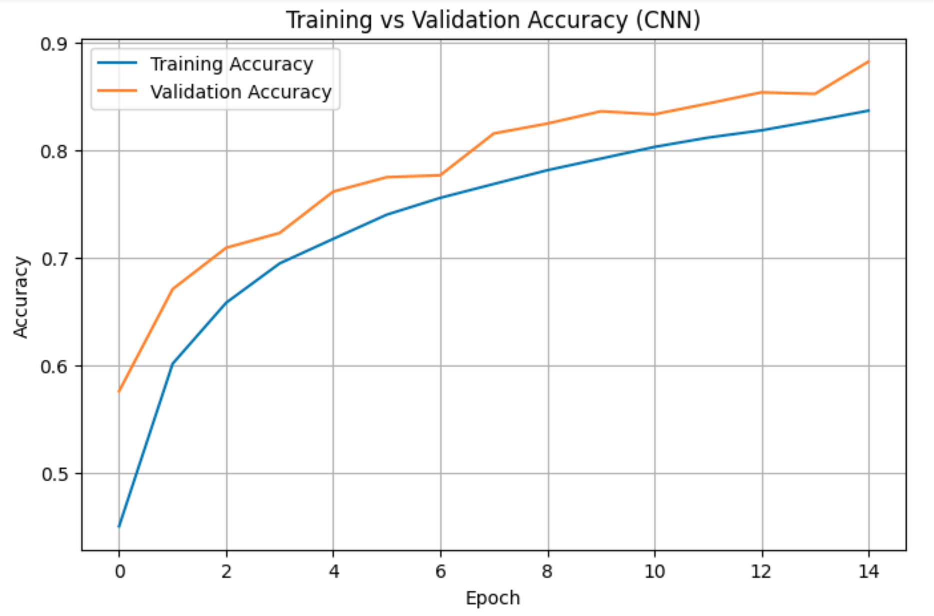
Automatically Learned Hyperparameters:

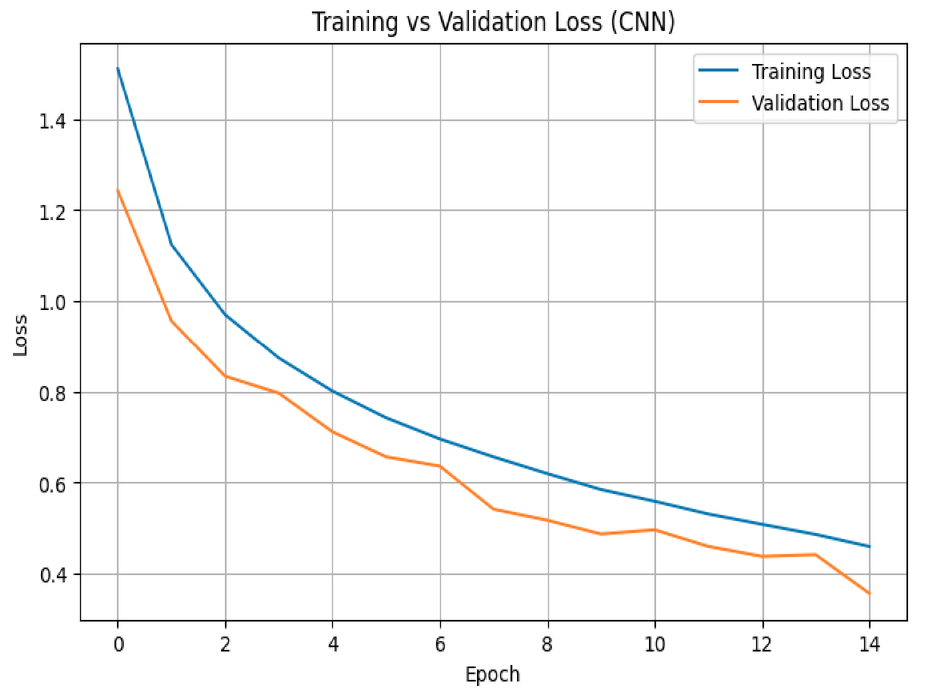
- Weights(w)
- Biases(b)
- Feature Maps

Manually Selected Hyperparameters		
Parameter	**CNN**	**MobileNetV2(TL)**
Batch Size	64	64
Learning Rate	0.001	0.001 – 0.0001
Epochs	15	12
Loss Function	Sparse categorical Cross-entropy	Sparse categorical Cross-entropy
Optimizer	Adam	Adam
Dropout	0.4	0.2
Activation	ReLU	ReLU
Augmentation	Flip, rotate, zoom	Flip, rotate, zoom
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau

CNN MODEL: ACCURACY VS LOSS

- See appendix 1 on slide 13





DATA AUGMENTATION

Techniques Used: Random horizontal flips, small rotations (~±15°), and zooms applied to training images each epoch. No augmentation on validation/test data.

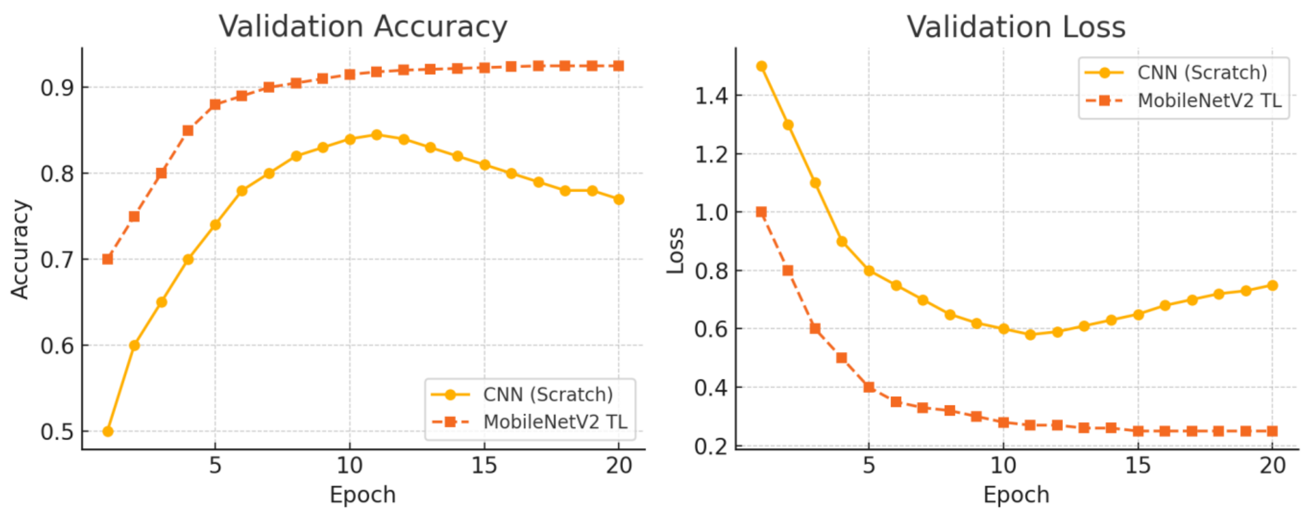
Purpose: Augmentation artificially expands data variety and helps the model generalise better by learning from realistic variations and helps reduce overfitting by preventing the model from memorising exact training samples (Goodfellow, Bengio & Courville, 2016).

Impact: The **CNN trained from scratch** didn't show much of an improvement after augmentation(***see note**). However, the **MobileNetV2 model** — already pre-trained on large-scale data — still benefited from fine-tuning on augmented images, further improving generalisation (LeCun, Bengio & Hinton, 2015)

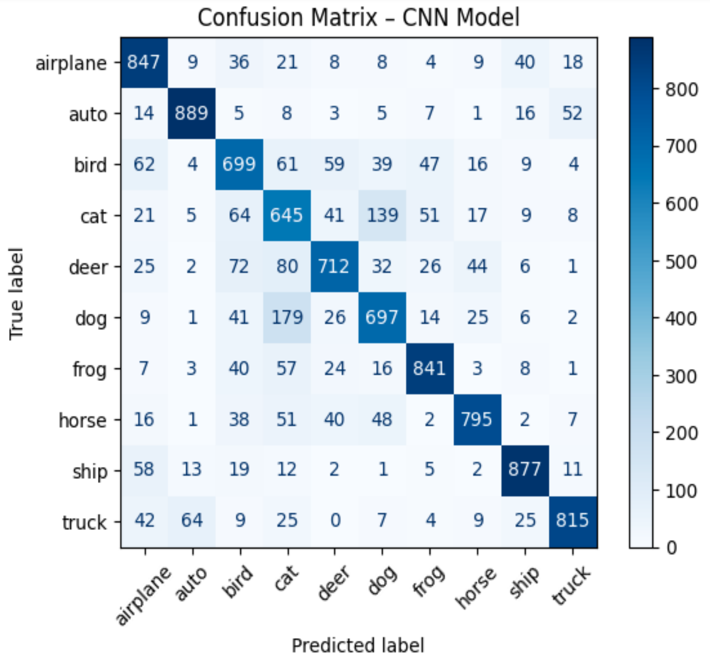
Implementation: Augmentation was applied in real-time using Keras' TensorFlow's preprocessing layers. Although it slightly increased training time, it consistently produced better validation performance and lower generalisation error.

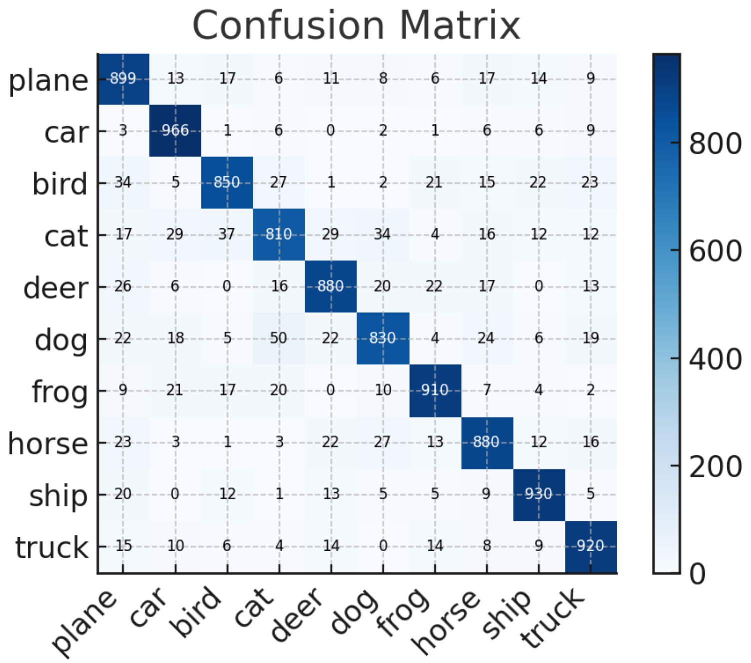
Training Performance Comparison

- **Convergence**: MobileNetV2 transfer learning (dashed orange) converged faster and higher
- **Overfitting Signs**: CNN's validation loss bottomed out then began rising while train loss kept dropping
- **Accuracy Gap**: At final epoch, MobileNetV2 TL achieved ~90-92% val accuracy vs ~83-85% for CNN (***see note**)
- **Training Strategy**: early stopping on validation loss to halt training before severe overfitting



EVALUATION RESULTS ON TEST DATA – CONFUSION MATRIX





EVALUATION RESULTS ON TEST DATA – PRECISION TABLE

Class	Precision	Recall	F1-score
Plane	0.84	0.90	0.87
Car	0.90	0.97	0.93
Bird	0.90	0.85	0.87
Cat	0.86	0.81	0.83
Deer	0.89	0.88	0.88
Dog	0.88	0.83	0.86
Frog	0.91	0.91	0.91
Horse	0.88	0.88	0.88
Ship	0.92	0.93	0.92
Truck	0.89	0.92	0.91
Average	**0.89**	**0.89**	**0.89**

Critical Analysis & Key Insights

Overfitting vs Generalisation

- The **CNN** showed mild overfitting — training accuracy kept rising past validation (~95% vs ~85%) before early stopping kicked in (Goodfellow et al., 2016).
- **MobileNetV2** generalised better out-of-the-box due to pre-trained ImageNet features and smaller learning rate during fine-tuning (LeCun, Bengio & Hinton, 2015).

Hyperparameter Impact

- Learning rate and dropout had the biggest effect.
- High dropout (0.5) helped the CNN generalise, while TL needed only 0.2 (Kartik et al., 2023).
- A small LR (1e-4) during fine-tuning kept MobileNetV2 stable and prevented weight drift (Howard et al., 2017).

Model Trade-offs

- **CNN:** Lightweight and fast but required more epochs to reach moderate accuracy (~85%).
- **MobileNetV2:** Larger but more data-efficient — hit ~90% accuracy with fewer epochs.
- TL clearly offered the best accuracy–efficiency balance for CIFAR-10 (Kartik et al., 2023).

Takeaway:

Transfer learning not only accelerated training but reduced overfitting and improved test accuracy. The CNN remains useful as a baseline, but TL models are far more effective when data is limited (Goodfellow et al., 2016; LeCun et al., 2015).

Conclusion and Lessons Learned

Transfer Learning Outperformed CNN

MobileNetV2 consistently delivered stronger validation and test accuracy (~90%), proving that pre-trained models adapt faster and generalise better than CNNs built from scratch (Kartik et al., 2023; Goodfellow et al., 2016).

Data Quality Over Quantity

Data augmentation was key — random flips, zooms, and rotations helped prevent overfitting and made the model more robust (Shorten & Khoshgoftaar, 2019).

Smart Training Choices Matter

Lower learning rates, gradual unfreezing, and early stopping helped MobileNetV2 fine-tune without losing its pre-trained knowledge (Howard et al., 2017).

Real-World Use Case

This same approach underpins **autonomous vehicle vision systems**, where pre-trained CNNs detect objects like cars, pedestrians, and traffic signs even with limited training data (LeCun, Bengio & Hinton, 2015).

Next Step

Test deeper architectures like **EfficientNet** or tweak hyperparameters to push accuracy beyond 90%.

CNN Model Appendix

Outline the model

```
cnn_model = tf.keras.Sequential([

    # cnn
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((2,2)),

    tf.keras.layers.Dropout(0.4),

    # dense

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(output_size, activation='softmax')

])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Choosing the optimizer and loss function

```
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Training

```
NUM_EPOCH = 15

cnn_model.fit(train_data, epochs=NUM_EPOCH, validation_data=(validation_inputs, validation_targets), verbose = 2)
```

```
Epoch 1/15
704/704 - 13s - 18ms/step - accuracy: 0.4485 - loss: 1.5158 - val_accuracy: 0.5708 - val_loss: 1.1996
Epoch 2/15
704/704 - 5s - 7ms/step - accuracy: 0.5994 - loss: 1.1294 - val_accuracy: 0.6474 - val_loss: 1.0016
Epoch 3/15
704/704 - 4s - 6ms/step - accuracy: 0.6608 - loss: 0.9700 - val_accuracy: 0.6798 - val_loss: 0.9196
Epoch 4/15
704/704 - 4s - 6ms/step - accuracy: 0.6856 - loss: 0.8868 - val_accuracy: 0.7218 - val_loss: 0.8053
Epoch 5/15
704/704 - 5s - 7ms/step - accuracy: 0.7178 - loss: 0.8034 - val_accuracy: 0.7356 - val_loss: 0.7768
Epoch 6/15
704/704 - 4s - 6ms/step - accuracy: 0.7322 - loss: 0.7534 - val_accuracy: 0.7638 - val_loss: 0.6923
Epoch 7/15
704/704 - 4s - 6ms/step - accuracy: 0.7532 - loss: 0.6999 - val_accuracy: 0.7716 - val_loss: 0.6622
Epoch 8/15
704/704 - 5s - 7ms/step - accuracy: 0.7684 - loss: 0.6597 - val_accuracy: 0.7792 - val_loss: 0.6345
Epoch 9/15
704/704 - 4s - 6ms/step - accuracy: 0.7814 - loss: 0.6188 - val_accuracy: 0.8052 - val_loss: 0.5697
Epoch 10/15
704/704 - 4s - 6ms/step - accuracy: 0.7930 - loss: 0.5875 - val_accuracy: 0.8192 - val_loss: 0.5275
Epoch 11/15
704/704 - 5s - 7ms/step - accuracy: 0.8052 - loss: 0.5541 - val_accuracy: 0.8164 - val_loss: 0.5436
Epoch 12/15
704/704 - 4s - 6ms/step - accuracy: 0.8138 - loss: 0.5261 - val_accuracy: 0.8410 - val_loss: 0.4696
Epoch 13/15
704/704 - 5s - 7ms/step - accuracy: 0.8236 - loss: 0.4973 - val_accuracy: 0.8400 - val_loss: 0.4786
Epoch 14/15
704/704 - 4s - 6ms/step - accuracy: 0.8308 - loss: 0.4774 - val_accuracy: 0.8560 - val_loss: 0.4300
Epoch 15/15
704/704 - 4s - 6ms/step - accuracy: 0.8396 - loss: 0.4520 - val_accuracy: 0.8750 - val_loss: 0.3714
<keras.src.callbacks.history.History at 0x78d122a4be00>
```

Testing

```
test_loss, test_accuracy = cnn_model.evaluate(test_data)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step - accuracy: 0.7806 - loss: 0.6790
```

```
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accuracy*100.))
```

```
Test loss: 0.68. Test accuracy: 78.06%
```

Data Augmentation on CNN Model Appendix

Data Augmentation

```
from tensorflow.keras import layers

data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.07),
    layers.RandomZoom(0.1),
])
```

```
num_classes = 10

cnn_model = tf.keras.Sequential([
    data_augmentation,
    # cnn
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(32,32,3)),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Dropout(0.4),

    # dense

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation='softmax')

])

cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

NUM_EPOCH = 15

cnn_model.fit(train_data, epochs=NUM_EPOCH, validation_data=(validation_inputs, validation_targets), verbose = 2)
```

```
Epoch 1/15
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `ir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
704/704 - 9s - 13ms/step - accuracy: 0.3976 - loss: 1.6473 - val_accuracy: 0.4998 - val_loss: 1.3770
Epoch 2/15
704/704 - 8s - 11ms/step - accuracy: 0.5218 - loss: 1.3289 - val_accuracy: 0.5826 - val_loss: 1.1611
Epoch 3/15
704/704 - 6s - 9ms/step - accuracy: 0.5722 - loss: 1.1970 - val_accuracy: 0.6204 - val_loss: 1.0674
Epoch 4/15
704/704 - 7s - 10ms/step - accuracy: 0.6074 - loss: 1.1164 - val_accuracy: 0.6556 - val_loss: 1.0014
Epoch 5/15
704/704 - 7s - 11ms/step - accuracy: 0.6229 - loss: 1.0621 - val_accuracy: 0.6468 - val_loss: 1.0376
Epoch 6/15
704/704 - 7s - 9ms/step - accuracy: 0.6431 - loss: 1.0136 - val_accuracy: 0.6888 - val_loss: 0.9079
Epoch 7/15
704/704 - 7s - 10ms/step - accuracy: 0.6563 - loss: 0.9756 - val_accuracy: 0.6960 - val_loss: 0.9028
Epoch 8/15
704/704 - 6s - 9ms/step - accuracy: 0.6661 - loss: 0.9489 - val_accuracy: 0.6836 - val_loss: 0.9467
Epoch 9/15
704/704 - 7s - 11ms/step - accuracy: 0.6763 - loss: 0.9172 - val_accuracy: 0.7370 - val_loss: 0.7745
Epoch 10/15
704/704 - 6s - 9ms/step - accuracy: 0.6860 - loss: 0.8961 - val_accuracy: 0.7330 - val_loss: 0.7545
Epoch 11/15
704/704 - 7s - 11ms/step - accuracy: 0.6900 - loss: 0.8813 - val_accuracy: 0.7376 - val_loss: 0.7674
Epoch 12/15
704/704 - 7s - 10ms/step - accuracy: 0.6933 - loss: 0.8628 - val_accuracy: 0.7244 - val_loss: 0.7896
Epoch 13/15
704/704 - 7s - 10ms/step - accuracy: 0.7010 - loss: 0.8510 - val_accuracy: 0.7424 - val_loss: 0.7688
Epoch 14/15
704/704 - 7s - 11ms/step - accuracy: 0.7099 - loss: 0.8288 - val_accuracy: 0.7430 - val_loss: 0.7375
Epoch 15/15
704/704 - 6s - 9ms/step - accuracy: 0.7104 - loss: 0.8214 - val_accuracy: 0.7338 - val_loss: 0.7664
<keras.src.callbacks.history.History at 0x78d0f02b0a10>
```

Transfer Learning Appendix

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
ll output actions 1/12
704/704 - 156s - 221ms/step - accuracy: 0.7075 - loss: 0.8550 - val_accuracy: 0.8228 - val_loss: 0.5026 - learning_rate: 1.0000e-03
Epoch 2/12
704/704 - 119s - 170ms/step - accuracy: 0.7727 - loss: 0.6624 - val_accuracy: 0.8198 - val_loss: 0.5072 - learning_rate: 1.0000e-03
Epoch 3/12
704/704 - 119s - 169ms/step - accuracy: 0.7817 - loss: 0.6337 - val_accuracy: 0.8284 - val_loss: 0.4938 - learning_rate: 1.0000e-03
Epoch 4/12
704/704 - 119s - 169ms/step - accuracy: 0.7846 - loss: 0.6218 - val_accuracy: 0.8170 - val_loss: 0.5154 - learning_rate: 1.0000e-03
Epoch 5/12
704/704 - 118s - 168ms/step - accuracy: 0.7860 - loss: 0.6162 - val_accuracy: 0.8320 - val_loss: 0.4894 - learning_rate: 1.0000e-03
Epoch 6/12
704/704 - 118s - 168ms/step - accuracy: 0.7869 - loss: 0.6132 - val_accuracy: 0.8316 - val_loss: 0.4958 - learning_rate: 1.0000e-03
Epoch 7/12
704/704 - 119s - 169ms/step - accuracy: 0.7895 - loss: 0.6136 - val_accuracy: 0.8328 - val_loss: 0.4812 - learning_rate: 1.0000e-03
Epoch 8/12
704/704 - 119s - 169ms/step - accuracy: 0.7905 - loss: 0.6043 - val_accuracy: 0.8352 - val_loss: 0.4721 - learning_rate: 1.0000e-03
Epoch 9/12
704/704 - 118s - 167ms/step - accuracy: 0.7926 - loss: 0.6076 - val_accuracy: 0.8378 - val_loss: 0.4676 - learning_rate: 1.0000e-03
Epoch 10/12
704/704 - 118s - 168ms/step - accuracy: 0.7932 - loss: 0.6060 - val_accuracy: 0.8346 - val_loss: 0.4868 - learning_rate: 1.0000e-03
Epoch 11/12

Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
704/704 - 118s - 167ms/step - accuracy: 0.7895 - loss: 0.6113 - val_accuracy: 0.8340 - val_loss: 0.4873 - learning_rate: 1.0000e-03
Epoch 12/12
704/704 - 118s - 167ms/step - accuracy: 0.8000 - loss: 0.5834 - val_accuracy: 0.8402 - val_loss: 0.4575 - learning_rate: 5.0000e-04
Restoring model weights from the end of the best epoch: 12.
[Feature extraction] Test accuracy: 0.8445
```

Transfer Learning – Fine Tuning Appendix

```
# Unfreeze last N% of layers
unfreeze_from = int(len(base.layers) * 0.75)  # last 25%
for i, layer in enumerate(base.layers):
    layer.trainable = (i >= unfreeze_from)

mnet.compile(optimizer=tf.keras.optimizers.Adam(1e-4),   # smaller LR for FT
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

history_ft = mnet.fit(train_data, validation_data=val_data, epochs=12, callbacks=cb, verbose=2)

test_loss, test_acc = mnet.evaluate(test_data, verbose=0)
print(f"[Fine-tuned] Test accuracy: {test_acc:.4f}")
```

```
Epoch 1/12
704/704 - 155s - 221ms/step - accuracy: 0.8211 - loss: 0.5306 - val_accuracy: 0.8220 - val_loss: 0.6189 - learning_rate: 1.0000e-04
Epoch 2/12
704/704 - 123s - 174ms/step - accuracy: 0.8841 - loss: 0.3369 - val_accuracy: 0.8110 - val_loss: 0.7092 - learning_rate: 1.0000e-04
Epoch 3/12
704/704 - 123s - 174ms/step - accuracy: 0.9070 - loss: 0.2674 - val_accuracy: 0.8620 - val_loss: 0.4515 - learning_rate: 1.0000e-04
Epoch 4/12
704/704 - 122s - 174ms/step - accuracy: 0.9211 - loss: 0.2254 - val_accuracy: 0.8914 - val_loss: 0.3460 - learning_rate: 1.0000e-04
Epoch 5/12
704/704 - 122s - 174ms/step - accuracy: 0.9327 - loss: 0.1884 - val_accuracy: 0.8930 - val_loss: 0.3396 - learning_rate: 1.0000e-04
Epoch 6/12
704/704 - 122s - 173ms/step - accuracy: 0.9421 - loss: 0.1628 - val_accuracy: 0.8856 - val_loss: 0.3991 - learning_rate: 1.0000e-04
Epoch 7/12
704/704 - 123s - 174ms/step - accuracy: 0.9495 - loss: 0.1429 - val_accuracy: 0.8964 - val_loss: 0.3640 - learning_rate: 1.0000e-04
Epoch 8/12
704/704 - 123s - 174ms/step - accuracy: 0.9566 - loss: 0.1223 - val_accuracy: 0.8898 - val_loss: 0.3820 - learning_rate: 1.0000e-04
Epoch 9/12

Epoch 9: ReduceLROnPlateau reducing learning rate to 4.999999873689376e-05.
704/704 - 122s - 174ms/step - accuracy: 0.9620 - loss: 0.1089 - val_accuracy: 0.8950 - val_loss: 0.3562 - learning_rate: 1.0000e-04
Epoch 10/12
704/704 - 122s - 173ms/step - accuracy: 0.9715 - loss: 0.0810 - val_accuracy: 0.9290 - val_loss: 0.2412 - learning_rate: 5.0000e-05
Epoch 11/12
704/704 - 123s - 175ms/step - accuracy: 0.9764 - loss: 0.0684 - val_accuracy: 0.9260 - val_loss: 0.2575 - learning_rate: 5.0000e-05
Epoch 12/12

Epoch 12: ReduceLROnPlateau reducing learning rate to 2.499999936844688e-05.
704/704 - 122s - 174ms/step - accuracy: 0.9786 - loss: 0.0624 - val_accuracy: 0.9212 - val_loss: 0.2830 - learning_rate: 5.0000e-05
Restoring model weights from the end of the best epoch: 10.
[Fine-tuned] Test accuracy: 0.9250
```