In the modified version of my Convolutional Neural Network (CNN), I made several improvements to increase its performance and generalization compared to the original baseline model:

1. Added Batch Normalization:
    - Batch normalization layers were added after convolution layers to stabilize training, speed up convergence, and reduce sensitivity to weight initialization.
    - This helped the model train more efficiently and improved overall accuracy.
2. Introduced Dropout Layers:
    - Dropout was added before the dense layer to prevent overfitting by randomly deactivating a portion of neurons during training.
    - This forced the model to learn more robust and generalized feature representations.
3. Increased Dense Layer Capacity:
    - The fully connected layer was increased from 64 to 128 units to improve the model's ability to learn complex decision boundaries.
    - A larger dense layer allows the network to combine learned features more effectively before final classification.
4. Added a Softmax Activation to the Output Layer:
    - The original output layer had no activation, so I added a softmax layer to convert raw logits into probability distributions.
    - This improved the model's stability and interpretability during training and evaluation.

```
import tensorflow as tf
import numpy as np
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

```
2025-10-07 08:33:40.226959: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plug
in cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1759826020.461429      36 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been
registered
E0000 00:00:1759826020.524556      36 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already be
en registered
```

+ Code + Markdown

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
```

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

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

```
model = models.Sequential()

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

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

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())


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

model.add(layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 128)	131,200
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 188,810 (737.54 KB)
Trainable params: 188,810 (737.54 KB)
Non-trainable params: 0 (0.00 B)

+ Code + Markdown

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

[10]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 38s 23ms/step - accuracy: 0.7204 - loss: 0.7926 - val_accuracy: 0.6897 - val_loss: 0.9113
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7339 - loss: 0.7615 - val_accuracy: 0.7074 - val_loss: 0.8591
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 22ms/step - accuracy: 0.7388 - loss: 0.7371 - val_accuracy: 0.7081 - val_loss: 0.8576
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7438 - loss: 0.7194 - val_accuracy: 0.7138 - val_loss: 0.8840
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7584 - loss: 0.6904 - val_accuracy: 0.7213 - val_loss: 0.8517
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7615 - loss: 0.6713 - val_accuracy: 0.7002 - val_loss: 0.9072
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7695 - loss: 0.6384 - val_accuracy: 0.7131 - val_loss: 0.8825
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7772 - loss: 0.6256 - val_accuracy: 0.7084 - val_loss: 0.9092
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 23ms/step - accuracy: 0.7831 - loss: 0.5985 - val_accuracy: 0.7128 - val_loss: 0.9051
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 35s 22ms/step - accuracy: 0.7901 - loss: 0.5869 - val_accuracy: 0.7097 - val_loss: 0.9025
```

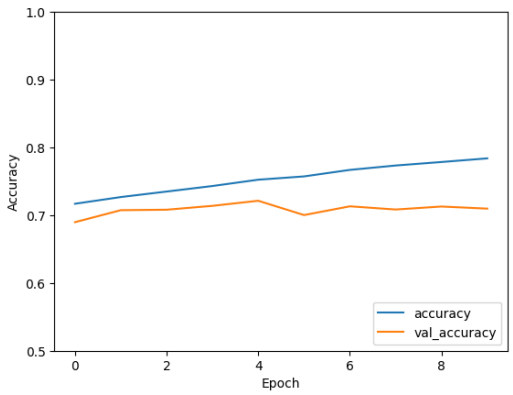
+ Code + Markdown

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 2s - 7ms/step - accuracy: 0.7097 - loss: 0.9025
```

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 2s - 7ms/step - accuracy: 0.7097 - loss: 0.9025
```



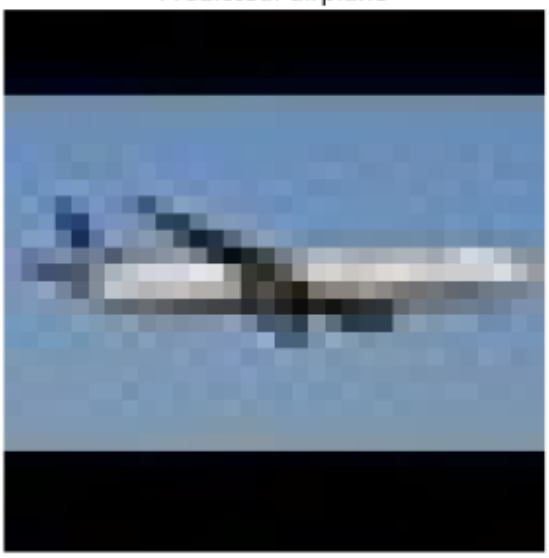
+ Code + Markdown

[9]:

```
from tensorflow.keras.preprocessing import image
import matplotlib.image as mpimg

image_paths = [ "/kaggle/input/my-images/airplane.jpg",
                "/kaggle/input/my-images/cat.jpg",
               "/kaggle/input/my-images/truck.jpeg" ]
for img_path in image_paths:
    img = image.load_img(img_path, target_size=(32, 32))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction[0])

    plt.imshow(img)
    plt.title(f"Predicted: {class_names[predicted_class]}")
    plt.axis("off")
    plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step
```

Predicted: airplane



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
```

Predicted: cat

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```

Predicted: truck