



# **Project I | Deep Learning: Image Classification with CNN**

**CIFAR-10 dataset**

**Nataliya Demyanenko**

# Project Overview

- Goal of the project: build a Convolutional Neural Network (CNN) model to classify images from the CIFAR-10 dataset into 10 predefined categories/classes
- CIFAR-10 dataset:
  - 60,000 32x32 color images
  - from 10 classes
  - 6,000 images per class





# Data Processing

- Split the data set into training, validation, and test sets, where 20% were allocated for validation.
- Normalized the image data by scaling
- Augmenting data using such techniques as rotation, shifting, shearing, zooming, and flipping
- Formatted the images and labels
- Prepared data generators to feed them into the model in batches
- One-hot encoded labels

# Basic Model

The model showed 0.553 accuracy on validation set and 0.547 accuracy on test set

```
#Define a cnn model
def build_cnn_model():
    cnn_model = tf.keras.Sequential([
        # First convolutional layer
        tf.keras.layers.Conv2D(16, (5, 5), strides=(1, 1), activation='relu', input_shape=(32, 32, 3)),

        # First max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),

        # Second convolutional layer
        tf.keras.layers.Conv2D(8, (5, 5), strides=(1, 1), activation='relu'),

        # Second max pooling layer
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),

        # Output layer
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    return cnn_model
```



## Steps that didn't help

- Increasing the number of epochs to 100,
- Setting learning rate to 0.001,
- Setting learning rate to 0.0001,
- Changing optimizer to RMSprop,
- Adding another convolutional layer,
- Adding another dense layer.

# Better Models

	Doubled number of filters and neurons in the model with three convolutional layers	Added 3d MaxPool layer and batch normalizations after every layer	Benchmark from the web	InceptionV3	VGG16	VGG16 fine tuned
Validation accuracy	0.632	0.666	0.818	0.643	0.697	0.591
Test accuracy	0.636	0.686	0.817	0.639	0.692	0.524
Precision	0.635	0.688	0.819	0.645	0.703	0.580
Recall	0.636	0.686	0.817	0.639	0.693	0.524
F1	0.633	0.685	0.815	0.638	0.691	0.532

# Selected Model

The architecture demonstrated the highest performance on all metrics and was selected for deployment.

```
#Define a cnn model
def build_cnn_model7():
    cnn_model7 = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu', input_shape=(32, 32, 3)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),
        tf.keras.layers.Dropout(0.3),

        tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1), activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1), activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1), activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(128, (3, 3), strides=(1, 1), activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)),
        tf.keras.layers.Dropout(0.5),

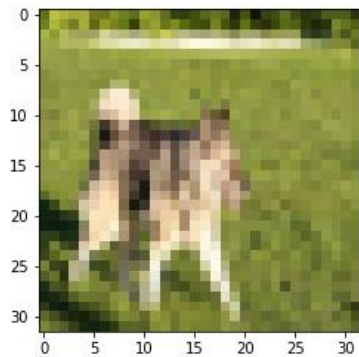
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation=tf.nn.relu),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.5),
        # Output layer
        tf.keras.layers.Dense(10, activation='softmax')

    ])

    return cnn_model7
```

# Local Deployment

- I deployed the best performing image classification model locally using Flask and TensorFlow Serving
- The models correctly classified the following images:



- The model incorrectly classified the following image as 'dog':






# Deployment on AWS

I took the following steps to deploy the model on AWS:

- Containerized my model into a Docker container;
- Set up account on AWS and launched and configured an EC2 instance;
- Connected to EC2 instance using PuTTY;
- Loaded the files to EC2 instance using Win SCP;
- Installed Docker on EC2;
- Encountered error when starting a container based on image on EC2 instance:

*docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "tensorflow\_model\_server": executable file not found in \$PATH: unknown.*

- A solution proposed on Stackflow suggests creating a separate Dockerfile that only runs the TensorFlow server, which I could not complete in time.



# Personal Takeaways

- Monitoring data input and output shape is important
- Caution in situations of exceptionally good model performance



**Thank you!**