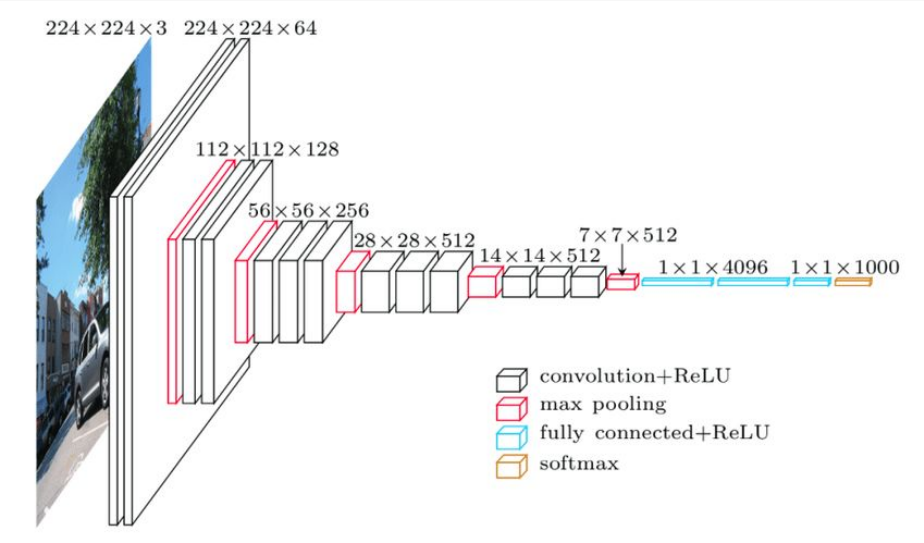
# CNN com Transfer Learning

# Modelo VGG16



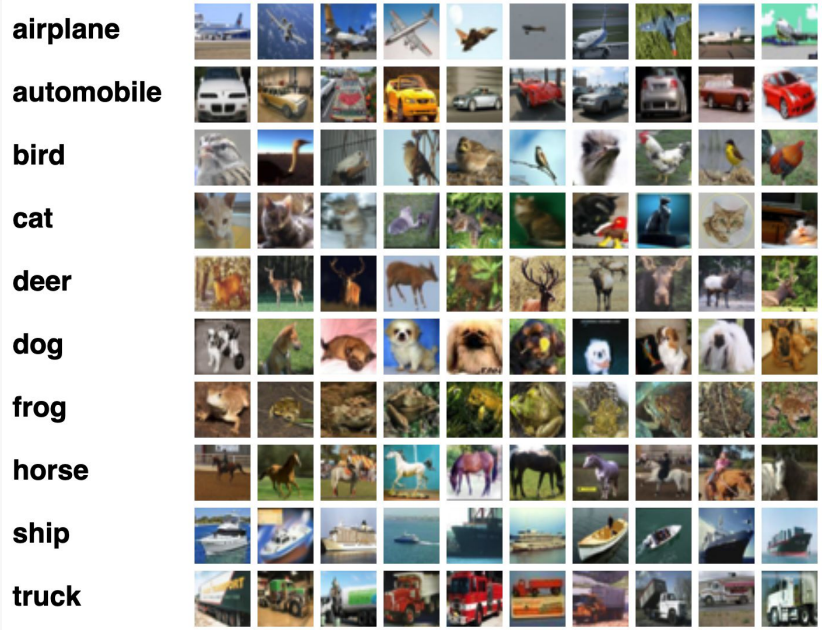

- O modelo VGG16 é uma arquitetura de CNN para tarefas de visão computacional. Desenvolvido pelo Visual Graphics Group (VGG) e apresentada em 2014.
- **16 camadas:**
  - **13 Camadas Convolucionais** (com filtros pequenos de 3x3)
  - **3 Camadas Densas**
- **Função de Ativação:** ReLU.
- Entrada: **imagem 224x224.**
- **138 milhões de parâmetros.**

# Bibliotecas utilizadas

```
import tensorflow as tf
from keras.applications.vgg16 import VGG16
from keras.datasets import cifar10
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
```

# Carregamento das imagens

```
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)
```



# Carregando o modelo VGG16 pré-treinado

```
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
```

# Modificando as últimas camadas do modelo

```
vgg16.trainable = True
fine_tune_at = 15

for layer in vgg16.layers[:fine_tune_at]:
    layer.trainable = False
```

Modifica as camadas finais (a partir da 15)

# Modificando as camadas 15 e 16

```
x = Flatten()(vgg16.output)
x = Dense(512, activation='relu')(x)
x = Dense(10, activation='softmax')(x)
```



10 classes do conjunto
de dados

# Criando e compilando o modelo

```
model = Model(inputs=vgg16.input, outputs=x)
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

# Gerando imagens extras

```
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2
)
datagen.fit(train_images)
```

Isso serve para termos uma maior quantidade de imagens para treino. Fazer isso melhorou a acurácia em 20%.

# Treinando o modelo

```
model.fit(datagen.flow(train_images,
                       train_labels,
                       batch_size=64),
          epochs=20,
          validation_data=(test_images,
                           test_labels))
```

# Avaliando o modelo (acurácia de teste): 75.33%

```
loss, accuracy = model.evaluate(test_images, test_labels)
```

# Plotando o resultado (imagens aleatórias)

```
def plot_image_predictions(model, images, true_labels, num_images=10):
    # Usa as predições do modelo
    predictions = model.predict(images[:num_images])
    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = np.argmax(true_labels[:num_images], axis=1)

    # Plota as imagens e as predições
    plt.figure(figsize=(15, 5))
    for i in range(num_images):
        plt.subplot(2, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"True: {true_classes[i]}, Pred: {predicted_classes[i]}")
        plt.axis('off')
    plt.show()

plot_image_predictions(model, test_images, test_labels)
```

# Plotando o resultado (imagens aleatórias)