In [1]:
```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers.core import Dense, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.pooling import MaxPool2D
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

np.random.seed(42)
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

In [2]:
```
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 11s 0us/step
170508288/170498071 [==============================] - 11s 0us/step
```

In [4]:
```
class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']
fig = plt.figure(figsize=(8,3))
for i in range(num_classes):
    ax = fig.add_subplot(2, 5, 1 + i, xticks=[], yticks=[])
    idx = np.where(Y_train[:]==i)[0]
    features_idx = X_train[idx,::]
    img_num = np.random.randint(features_idx.shape[0])
    im = features_idx[img_num,::]
    ax.set_title(class_names[i])
    plt = sns.lineplot(features_idx[img_num,::], [1, 2, 3, 4])
    plt.imshow(im)
plt.show()
```


Initializing model with no normalization

In [5]:
```
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# We add our classifier
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Training the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 19799s 51s/step - loss: 7.4793 - accuracy: 0.4085 - val_loss: 1.2218 - val_accuracy: 0.5698
Epoch 2/5
391/391 [==============================] - 609s 2s/step - loss: 0.9260 - accuracy: 0.6808 - val_loss: 1.0981 - val_accuracy: 0.6252
Epoch 3/5
391/391 [==============================] - 601s 2s/step - loss: 0.5230 - accuracy: 0.8274 - val_loss: 1.1936 - val_accuracy: 0.6135
Epoch 4/5
391/391 [==============================] - 996s 3s/step - loss: 0.2119 - accuracy: 0.9395 - val_loss: 1.3307 - val_accuracy: 0.6414
Epoch 5/5
391/391 [==============================] - 1147s 3s/step - loss: 0.0719 - accuracy: 0.9826 - val_loss: 1.5635 - val_accuracy: 0.6434
313/313 [==============================] - 78.237ms/step - loss: 1.5635 - accuracy: 0.6434
Loss: 1.563
Accuracy: 0.643
```

Data Pre-processing and normalization

Normalization is only done on the training data!

In [7]:
```
# Centering the data
X_train_mean = np.mean(X_train, axis = 0)
X_train_cent = X_train - X_train_mean
# Scaling the data
X_train_std = np.std(X_train, axis = 0)
X_train_norm = X_train_cent / X_train_std
```

In [8]:
```
X_test_norm = (X_test - X_train_mean) / X_train_std
```

Running model again with data normalization

In [9]:
```
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# We add our classifier
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Training the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 468s 1s/step - loss: 5.7854 - accuracy: 0.4794 - val_loss: 1.1737 - val_accuracy: 0.5909
Epoch 2/5
391/391 [==============================] - 468s 1s/step - loss: 0.8229 - accuracy: 0.7225 - val_loss: 1.1012 - val_accuracy: 0.6172
Epoch 3/5
391/391 [==============================] - 468s 1s/step - loss: 0.3568 - accuracy: 0.8889 - val_loss: 1.2949 - val_accuracy: 0.6438
Epoch 4/5
391/391 [==============================] - 475s 1s/step - loss: 0.1098 - accuracy: 0.9719 - val_loss: 1.4494 - val_accuracy: 0.6329
Epoch 5/5
391/391 [==============================] - 468s 1s/step - loss: 0.0353 - accuracy: 0.9952 - val_loss: 1.6619 - val_accuracy: 0.6337
313/313 [==============================] - 19s 60ms/step - loss: 1.6619 - accuracy: 0.6337
Loss: 1.662
Accuracy: 0.634
```

I can see that this helped make the training accuracy better but the validation accuracy is still low. We will now try different regularization techniques to make it less overfit

Batch Normalization

In [10]:
```
# We import Batch Normalization layer
from tensorflow.keras.layers import BatchNormalization, Activation
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), input_shape=(32, 32, 3)))
model.add(Activation('relu'))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
# We include our classifier
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test))) # aqui deberiamos usar un conjunto distinto al de test!!!
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 562s 1s/step - loss: 1.5839 - accuracy: 0.5243 - val_loss: 1.4914 - val_accuracy: 0.5296
Epoch 2/5
391/391 [==============================] - 579s 1s/step - loss: 0.7281 - accuracy: 0.7551 - val_loss: 0.9749 - val_accuracy: 0.6635
Epoch 3/5
391/391 [==============================] - 575s 1s/step - loss: 0.3370 - accuracy: 0.9073 - val_loss: 0.9748 - val_accuracy: 0.6763
Epoch 4/5
391/391 [==============================] - 592s 2s/step - loss: 0.1875 - accuracy: 0.9659 - val_loss: 1.0163 - val_accuracy: 0.6749
Epoch 5/5
391/391 [==============================] - 606s 2s/step - loss: 0.0107 - accuracy: 0.9992 - val_loss: 1.0356 - val_accuracy: 0.6964
313/313 [==============================] - 20s 63ms/step - loss: 1.0356 - accuracy: 0.6964
Loss: 1.036
Accuracy: 0.696
```

I can see that our training accuracy is pretty much perfect and that our validation accuracy improved slightly but can still get better. Using this also did make the neural network get trained faster.

Lasso Regularization (L1)

In [11]:
```
# L1 Regularization
# Regularizer layer import
from keras.regularizers import l2
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Classifier inclusion
model.add(Flatten())
model.add(Dense(1024, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 638s 2s/step - loss: 3.6848 - accuracy: 0.4655 - val_loss: 1.4961 - val_accuracy: 0.5443
Epoch 2/5
391/391 [==============================] - 621s 2s/step - loss: 1.4383 - accuracy: 0.5723 - val_loss: 1.3997 - val_accuracy: 0.5718
Epoch 3/5
391/391 [==============================] - 618s 2s/step - loss: 1.3189 - accuracy: 0.6135 - val_loss: 1.3195 - val_accuracy: 0.5988
Epoch 4/5
391/391 [==============================] - 616s 2s/step - loss: 1.2477 - accuracy: 0.6356 - val_loss: 1.2346 - val_accuracy: 0.6471
Epoch 5/5
391/391 [==============================] - 619s 2s/step - loss: 1.2028 - accuracy: 0.6544 - val_loss: 1.1836 - val_accuracy: 0.6625
313/313 [==============================] - 55s 176ms/step - loss: 1.1836 - accuracy: 0.6625
Loss: 1.184
Accuracy: 0.663
```

This did not really help since the accuracy score is actually a little lower and the loss amount is similar

Ridge Regularization (L2)

In [12]:
```
# L2 Regularization
# Regularizer layer import
from keras.regularizers import l1
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Classifier inclusion
model.add(Flatten())
model.add(Dense(1024, activation='relu', kernel_regularizer=l1(0.01)))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 621s 2s/step - loss: 204.0518 - accuracy: 0.2429 - val_loss: 12.8222 - val_accuracy: 0.2622
Epoch 2/5
391/391 [==============================] - 620s 2s/step - loss: 12.5092 - accuracy: 0.3172 - val_loss: 12.3096 - val_accuracy: 0.3403
Epoch 3/5
391/391 [==============================] - 621s 2s/step - loss: 12.1554 - accuracy: 0.3561 - val_loss: 12.0529 - val_accuracy: 0.3698
Epoch 4/5
391/391 [==============================] - 621s 2s/step - loss: 12.0569 - accuracy: 0.3813 - val_loss: 12.0312 - val_accuracy: 0.3833
Epoch 5/5
391/391 [==============================] - 621s 2s/step - loss: 11.9843 - accuracy: 0.3942 - val_loss: 11.7613 - val_accuracy: 0.4090
313/313 [==============================] - 55s 176ms/step - loss: 11.7613 - accuracy: 0.4090
Loss: 11.761
Accuracy: 0.409
```

This had extremely low accuracy and a high loss when compared to the other regularization methods

Elastic Net Regularization (L1+L2)

In [13]:
```
# Elastic Net Regularization (L1 + L2)
# Regularizer layer import
from keras.regularizers import l1_l2
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Classifier inclusion
model.add(Flatten())
model.add(Dense(1024, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 744s 2s/step - loss: 205.2887 - accuracy: 0.2286 - val_loss: 12.7639 - val_accuracy: 0.2356
Epoch 2/5
391/391 [==============================] - 740s 2s/step - loss: 12.4869 - accuracy: 0.3035 - val_loss: 12.2153 - val_accuracy: 0.2946
Epoch 3/5
391/391 [==============================] - 742s 2s/step - loss: 11.9370 - accuracy: 0.3432 - val_loss: 11.8029 - val_accuracy: 0.3324
Epoch 4/5
391/391 [==============================] - 737s 2s/step - loss: 11.7663 - accuracy: 0.3641 - val_loss: 11.6211 - val_accuracy: 0.3630
Epoch 5/5
391/391 [==============================] - 734s 2s/step - loss: 11.5752 - accuracy: 0.3779 - val_loss: 11.4863 - val_accuracy: 0.3828
313/313 [==============================] - 85s 269ms/step - loss: 11.4863 - accuracy: 0.3828
Loss: 11.486
Accuracy: 0.383
```

Combining the two did not seem to help very much as the accuracy was still low with a high loss

Max Norm Constraints

In [17]:
```
# Regularizer layer import
from tensorflow.keras.constraint import max_norm
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Classifier inclusion
model.add(Flatten())
model.add(Dense(1024, activation='relu', kernel_constraint=max_norm(3.)))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 838s 2s/step - loss: 1.3981 - accuracy: 0.5050 - val_loss: 1.1485 - val_accuracy: 0.5964
Epoch 2/5
391/391 [==============================] - 818s 2s/step - loss: 0.9618 - accuracy: 0.6610 - val_loss: 0.9853 - val_accuracy: 0.6575
Epoch 3/5
391/391 [==============================] - 824s 2s/step - loss: 0.7680 - accuracy: 0.7363 - val_loss: 0.9401 - val_accuracy: 0.6755
Epoch 4/5
391/391 [==============================] - 827s 2s/step - loss: 0.5863 - accuracy: 0.8027 - val_loss: 0.9161 - val_accuracy: 0.6898
Epoch 5/5
391/391 [==============================] - 833s 2s/step - loss: 0.4137 - accuracy: 0.8655 - val_loss: 0.9547 - val_accuracy: 0.6882
313/313 [==============================] - 26s 84ms/step - loss: 0.9547 - accuracy: 0.6882
Loss: 0.955
Accuracy: 0.688
```

Using constraints was an interesting way to perform regularization. While the accuracy was slightly lower than with just the batch normalization, the loss was slightly lower. I can also see that it is not nearly as overfit as the other methods as the accuracy and validation accuracy are not nearly as far apart

Dropout regularization

In [18]:
```
# Dropout
# Dropout layer import
from keras.layers import Dropout
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
# Classifier inclusion
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 577s 1s/step - loss: 1.5625 - accuracy: 0.4491 - val_loss: 1.2677 - val_accuracy: 0.5531
Epoch 2/5
391/391 [==============================] - 559s 1s/step - loss: 1.2038 - accuracy: 0.5759 - val_loss: 1.0888 - val_accuracy: 0.6194
Epoch 3/5
391/391 [==============================] - 558s 1s/step - loss: 1.0997 - accuracy: 0.6402 - val_loss: 0.9473 - val_accuracy: 0.6713
Epoch 4/5
391/391 [==============================] - 555s 1s/step - loss: 0.9579 - accuracy: 0.7014 - val_loss: 0.8966 - val_accuracy: 0.6892
Epoch 5/5
391/391 [==============================] - 554s 1s/step - loss: 0.7296 - accuracy: 0.7478 - val_loss: 0.8675 - val_accuracy: 0.7018
313/313 [==============================] - 19s 60ms/step - loss: 0.8675 - accuracy: 0.7018
Loss: 0.867
Accuracy: 0.702
```

This was very effective as it was able to get the value accuracy to be greater than 70% and a steep drop-off in the loss

Max Norm + Dropout

In [19]:
```
# Dropout & Max Norm
# Dropout & Max Norm layers import
from keras.layers import Dropout
from tensorflow.keras.constraint import max_norm
# Initializing the model
model = Sequential()
# Defining a convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Dropout(0.25))
# Defining a second convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
# Defining a third convolutional layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
# Classifier inclusion
model.add(Flatten())
model.add(Dense(1024, activation='relu', kernel_constraint=max_norm(3.)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(learning_rate=0.0001, decay=1e-6),
              metrics=['accuracy'])
# Training the model
model.fit(X_train_norm, to_categorical(Y_train),
          batch_size=128,
          shuffle=True,
          epochs=5,
          validation_data=(X_test_norm, to_categorical(Y_test)))
# Evaluating the model
scores = model.evaluate(X_test_norm, to_categorical(Y_test))
print('Loss: %.3f' % scores[0])
print('Accuracy: %.3f' % scores[1])
```

```
Epoch 1/5
391/391 [==============================] - 879s 2s/step - loss: 1.5965 - accuracy: 0.4357 - val_loss: 1.2516 - val_accuracy: 0.5578
Epoch 2/5
391/391 [==============================] - 878s 2s/step - loss: 1.2036 - accuracy: 0.5739 - val_loss: 1.0769 - val_accuracy: 0.6225
Epoch 3/5
391/391 [==============================] - 874s 2s/step - loss: 1.0158 - accuracy: 0.6448 - val_loss: 0.9549 - val_accuracy: 0.6665
Epoch 4/5
391/391 [==============================] - 885s 2s/step - loss: 0.8619 - accuracy: 0.6998 - val_loss: 0.9144 - val_accuracy: 0.6817
Epoch 5/5
391/391 [==============================] - 881s 2s/step - loss: 0.7207 - accuracy: 0.7451 - val_loss: 0.8531 - val_accuracy: 0.7066
313/313 [==============================] - 19s 62ms/step - loss: 0.8531 - accuracy: 0.7066
Loss: 0.853
Accuracy: 0.707
```

This was very similar to Dropout wit a slightly smaller loss. This makes sense since earlier we established that max norm didn't make much of a difference

In [ ]:
```
```