```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tabulate import tabulate
layers = tf.keras.layers
```

The code block below defines a few helper functions to visualize the results. You do not need to touch them.

```
def plot_examples(X, Y, n=10):
    """ Plot the first n examples for each of the 10 classes in the CIFAR dataset X, Y
    fig, axes = plt.subplots(n, 10, figsize=(10, n))
    for l in range(10):
        axes[0, l].set_title(cifar10_labels[l], fontsize="smaller")
        m = np.squeeze(Y) == l  # boolean mask: True for all images of label l
        for i in range(n):
            image = X[m][i].astype("uint8")  # imshow expects uint8
            ax = axes[i, l]
            ax.imshow(image, origin="upper")
            ax.set(xticks=[], yticks=[])
    return fig, ax


def plot_prediction(X, Y, Y_predict):
    """
    Plot image X along with predicted probabilities Y_predict.
    X: CIFAR image, shape = (32, 32, 3)
    Y: CIFAR label, one-hot encoded, shape = (10)
    Y_predict: predicted probabilities, shape = (10)
    """
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4))

    # plot image
    ax1.imshow(X.astype("uint8"), origin="upper")
    ax1.set(xticks=[], yticks=[])

    # plot probabilities
    ax2.barh(np.arange(10), Y_predict, align="center")
    ax2.set(xlim=(0, 1), xlabel="Score", yticks=[])
    for i in range(10):
        c = "red" if (i == np.argmax(Y)) else "black"
        ax2.text(0.05, i, cifar10_labels[i].capitalize(), ha="left", va="center", colo



def plot_confusion(Y_true, Y_predict):
    """
    Plot confusion matrix
    Y_true:    array of true classifications (0-9), shape = (N)
    Y_predict: array of predicted classifications (0-9), shape = (N)
    """
```

```
    C = np.histogram2d(Y_true, Y_predict, bins=np.linspace(-0.5, 9.5, 11))[0]
    Cn = C / np.sum(C, axis=1)

    fig = plt.figure()
    plt.imshow(Cn, interpolation="nearest", vmin=0, vmax=1, cmap=plt.cm.YlGnBu)
    plt.colorbar()
    plt.xlabel("prediction")
    plt.ylabel("truth")
    plt.xticks(range(10), cifar10_labels, rotation="vertical")
    plt.yticks(range(10), cifar10_labels)
    for x in range(10):
        for y in range(10):
            plt.annotate("%i" % C[x, y], xy=(y, x), ha="center", va="center")
```

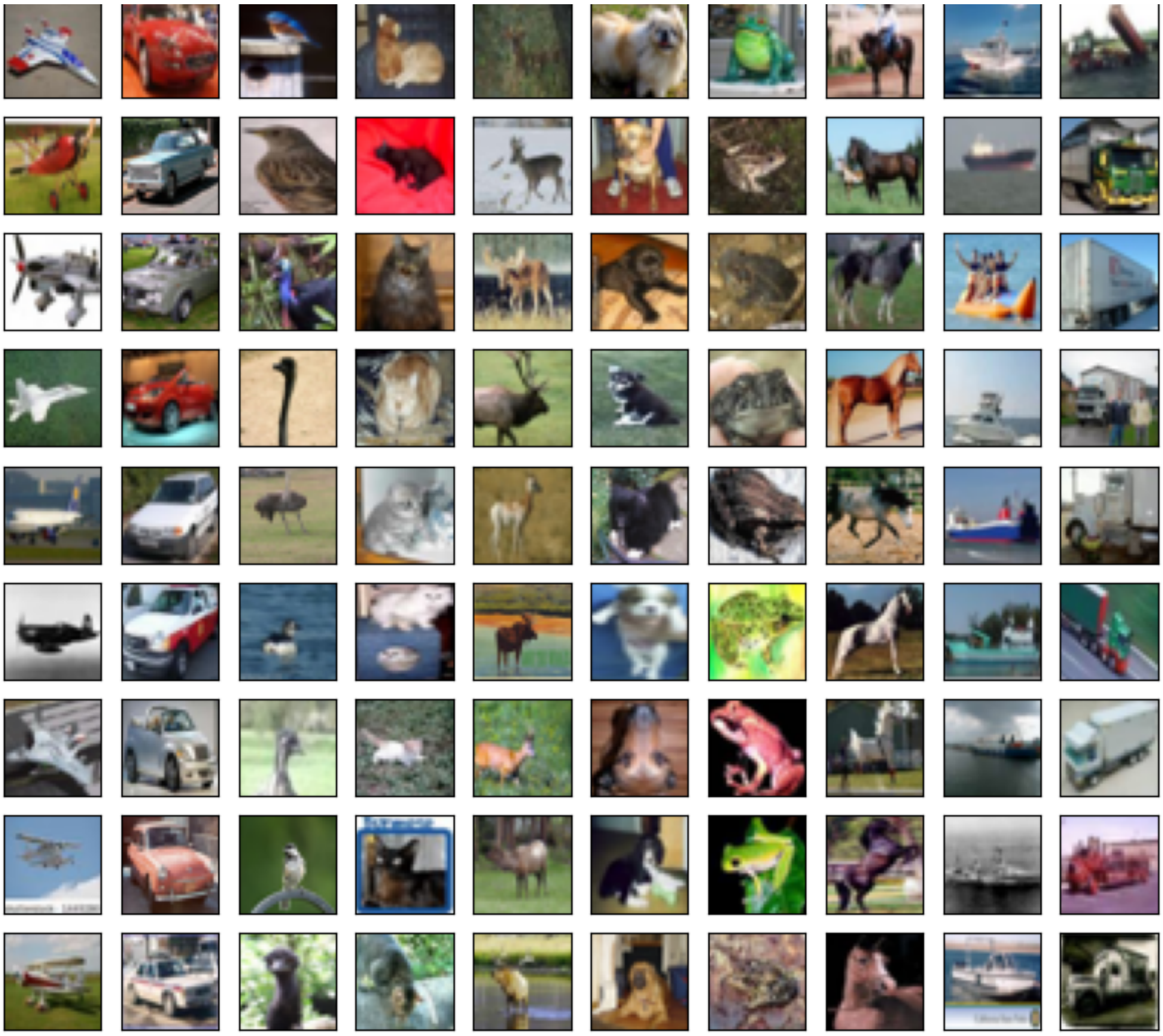
First we load and preprocess CIFAR-10 data. The imagages are 32x32 pixels and have three color channels (red, green blue).

```
# X: images, Y: labels
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

print("images, shape = ", x_train.shape)
print("labels, shape = ", y_train.shape)

cifar10_labels = np.array([
    'airplane',
    'automobile',
    'bird',
    'cat',
    'deer',
    'dog',
    'frog',
    'horse',
    'ship',
    'truck'])
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 12s 0us/step
images, shape =  (50000, 32, 32, 3)
labels, shape =  (50000, 1)
```

```
# Hint: To plot example images, you can use the plot examples function
plot_examples(x_train, y_train)
```

```
(<Figure size 1000x1000 with 100 Axes>, <Axes: >)
```



```
print(y_test.shape)
```

```
(10000, 1)
```

```
# convert labels ("0"-"9") to one-hot encodings, "0" = (1, 0, ... 0) and so on
y_train_onehot = tf.keras.utils.to_categorical(y_train, 10)
y_test_onehot = tf.keras.utils.to_categorical(y_test, 10)[:8000]
y_valid_onehot = tf.keras.utils.to_categorical(y_test, 10)[8000:]

# Hint: normalize the data
x_train_norm = (x_train-np.mean(x_train)) / np.std(x_train)
x_test_norm = (x_test-np.mean(x_test)) / np.std(x_test)

# Hint: use 20% of the training data for validation
```

```
x_valid_norm=x_test_norm[8000:]
x_test_norm=x_test_norm[:8000]
```

We start with a fully connected network

```
# --------------------------------------------------------------
# Define model
# --------------------------------------------------------------
model = tf.keras.models.Sequential(
    [
        layers.Flatten(input_shape=(32, 32, 3)),  # (32,32,3) --> (3072)
        # this time the flatten operation is directly integrated into the network
        # structure so that we can use the same input data later for a convolutional neu
        layers.Dense(256, activation="relu"),
        layers.Dropout(0.1),
        layers.Dense(256, activation="relu"),
        # Hint: remember that the output layer should have 10 nodes with a softmax activ
        layers.Dense(10, activation="softmax")
    ],
    name="nn",
)

print(model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37:
  super().__init__(**kwargs)
```

Model: "nn"

Layer (type)	Output Shape	Par
flatten (Flatten)	(None, 3072)	
dense (Dense)	(None, 256)	786
dropout (Dropout)	(None, 256)	
dense_1 (Dense)	(None, 256)	65
dense_2 (Dense)	(None, 10)	2

Total params: 855,050 (3.26 MB)
Trainable params: 855,050 (3.26 MB)
Non-trainable params: 0 (0.00 B)
None

```
# --------------------------------------------------------------
# Training
# --------------------------------------------------------------
model.compile(
    loss='categorical_crossentropy',
    optimizer="adam",
    metrics=["accuracy"]
)
```

```
model.fit(
    x_train_norm, y_train_onehot,
    batch_size=1000,
    epochs=20, # train at least for 20 epochs
    verbose=2,
    validation_data=(x_valid_norm, y_valid_onehot),
    callbacks=[tf.keras.callbacks.CSVLogger("history_{}.csv".format(model.name))],
)
```

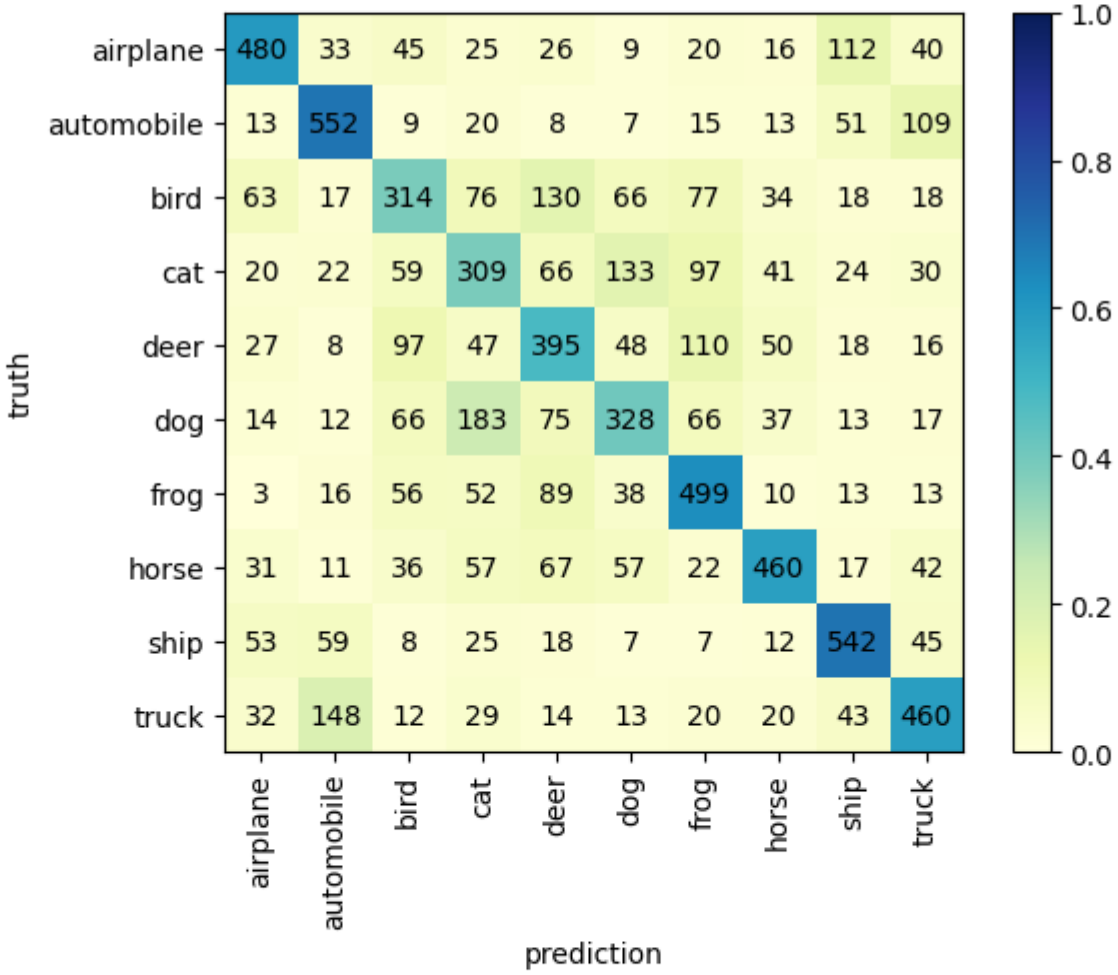
```
Epoch 1/20
50/50 - 6s - 124ms/step - accuracy: 0.3722 - loss: 1.8178 - val_accuracy: 0.4420 -
Epoch 2/20
50/50 - 4s - 80ms/step - accuracy: 0.4669 - loss: 1.5197 - val_accuracy: 0.4685 -
Epoch 3/20
50/50 - 7s - 137ms/step - accuracy: 0.5050 - loss: 1.4160 - val_accuracy: 0.4845 -
Epoch 4/20
50/50 - 4s - 80ms/step - accuracy: 0.5287 - loss: 1.3436 - val_accuracy: 0.4805 -
Epoch 5/20
50/50 - 4s - 82ms/step - accuracy: 0.5525 - loss: 1.2825 - val_accuracy: 0.5150 -
Epoch 6/20
50/50 - 6s - 110ms/step - accuracy: 0.5758 - loss: 1.2189 - val_accuracy: 0.5095 -
Epoch 7/20
50/50 - 4s - 80ms/step - accuracy: 0.5934 - loss: 1.1687 - val_accuracy: 0.5155 -
Epoch 8/20
50/50 - 6s - 113ms/step - accuracy: 0.6092 - loss: 1.1304 - val_accuracy: 0.5095 -
Epoch 9/20
50/50 - 5s - 99ms/step - accuracy: 0.6255 - loss: 1.0850 - val_accuracy: 0.5160 -
Epoch 10/20
50/50 - 4s - 82ms/step - accuracy: 0.6393 - loss: 1.0410 - val_accuracy: 0.5280 -
Epoch 11/20
50/50 - 6s - 127ms/step - accuracy: 0.6534 - loss: 0.9982 - val_accuracy: 0.5195 -
Epoch 12/20
50/50 - 9s - 179ms/step - accuracy: 0.6677 - loss: 0.9622 - val_accuracy: 0.5300 -
Epoch 13/20
50/50 - 7s - 133ms/step - accuracy: 0.6761 - loss: 0.9327 - val_accuracy: 0.5360 -
Epoch 14/20
50/50 - 9s - 177ms/step - accuracy: 0.6877 - loss: 0.8971 - val_accuracy: 0.5380 -
Epoch 15/20
50/50 - 6s - 111ms/step - accuracy: 0.6977 - loss: 0.8700 - val_accuracy: 0.5280 -
Epoch 16/20
50/50 - 4s - 80ms/step - accuracy: 0.7094 - loss: 0.8386 - val_accuracy: 0.5320 -
Epoch 17/20
50/50 - 5s - 95ms/step - accuracy: 0.7222 - loss: 0.8001 - val_accuracy: 0.5250 -
Epoch 18/20
50/50 - 5s - 96ms/step - accuracy: 0.7296 - loss: 0.7792 - val_accuracy: 0.5245 -
Epoch 19/20
50/50 - 4s - 87ms/step - accuracy: 0.7413 - loss: 0.7433 - val_accuracy: 0.5420 -
Epoch 20/20
50/50 - 7s - 131ms/step - accuracy: 0.7484 - loss: 0.7210 - val_accuracy: 0.5335 -
<keras.src.callbacks.history.History at 0x7babe10e6350>
```

Start coding or generate with AI.

```
# -----------------------------------------------------------
# Plots
# -----------------------------------------------------------
```

```
# training curves
history = np.genfromtxt("history_{}.csv".format(model.name), delimiter=",", names=True)


# Hint: this is how you can plot the confusion matrix.
# calculate predictions for test set
y_predict = model.predict(x_test_norm, batch_size=128)

# convert back to class labels (0-9)
y_predict_cl = np.argmax(y_predict, axis=1)
y_test_cl = np.argmax(y_test_onehot, axis=1)

# plot confusion matrix
plot_confusion(y_test_cl, y_predict_cl)

#
test_acc=np.mean(np.equal(y_predict_cl, y_test_cl))
print(test_acc)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
0.542375
```

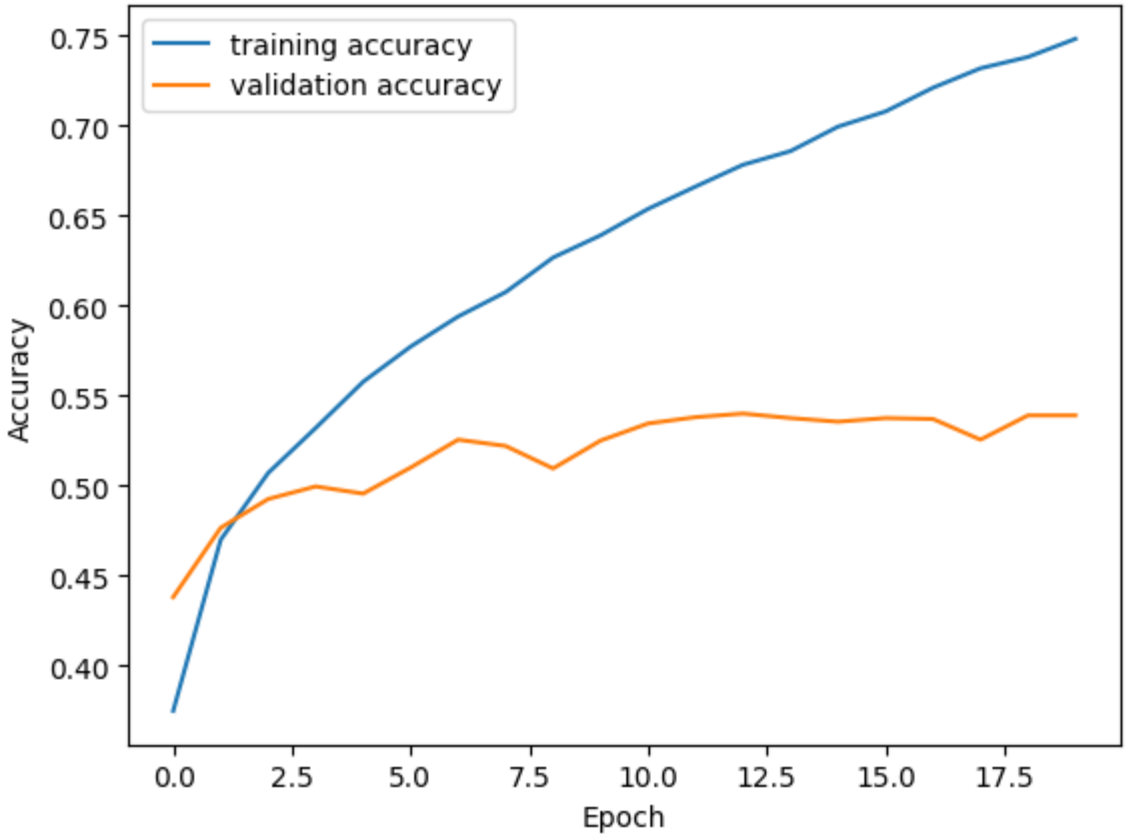


Start coding or generate with AI.

Start coding or generate with AI.

```
plt.figure()
plt.plot(history["epoch"], history["accuracy"], label="training accuracy")
plt.plot(history["epoch"], history["val_accuracy"], label="validation accuracy")
plt.legend()
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
```

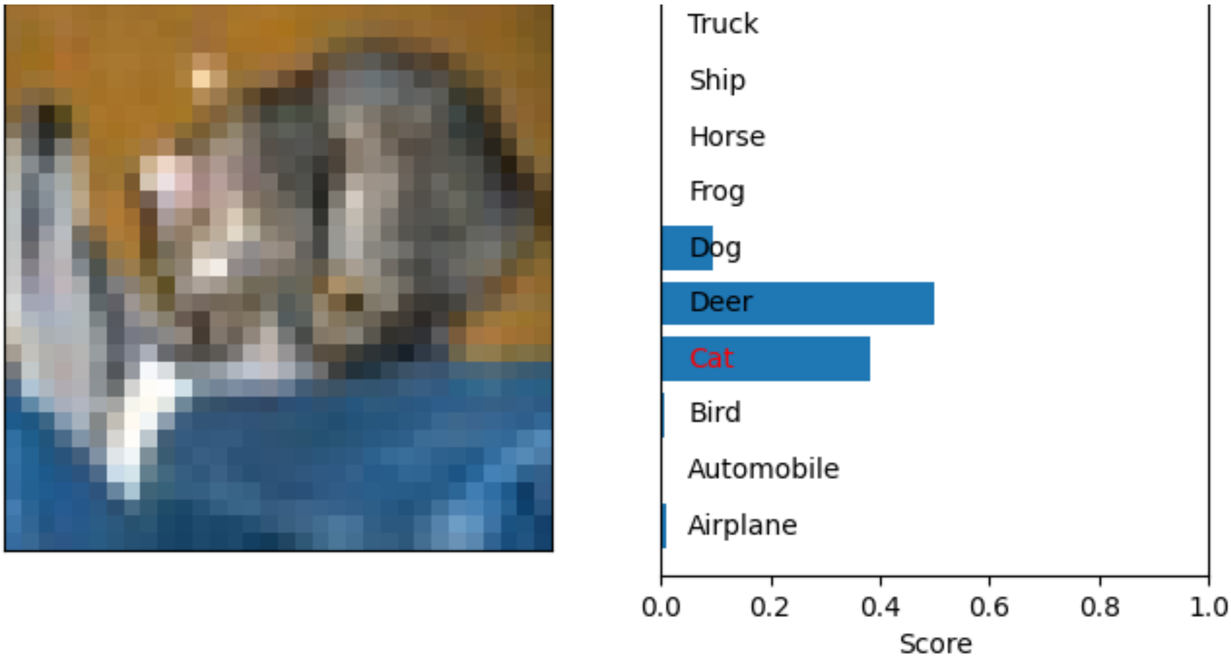
```
Text(0, 0.5, 'Accuracy')
```

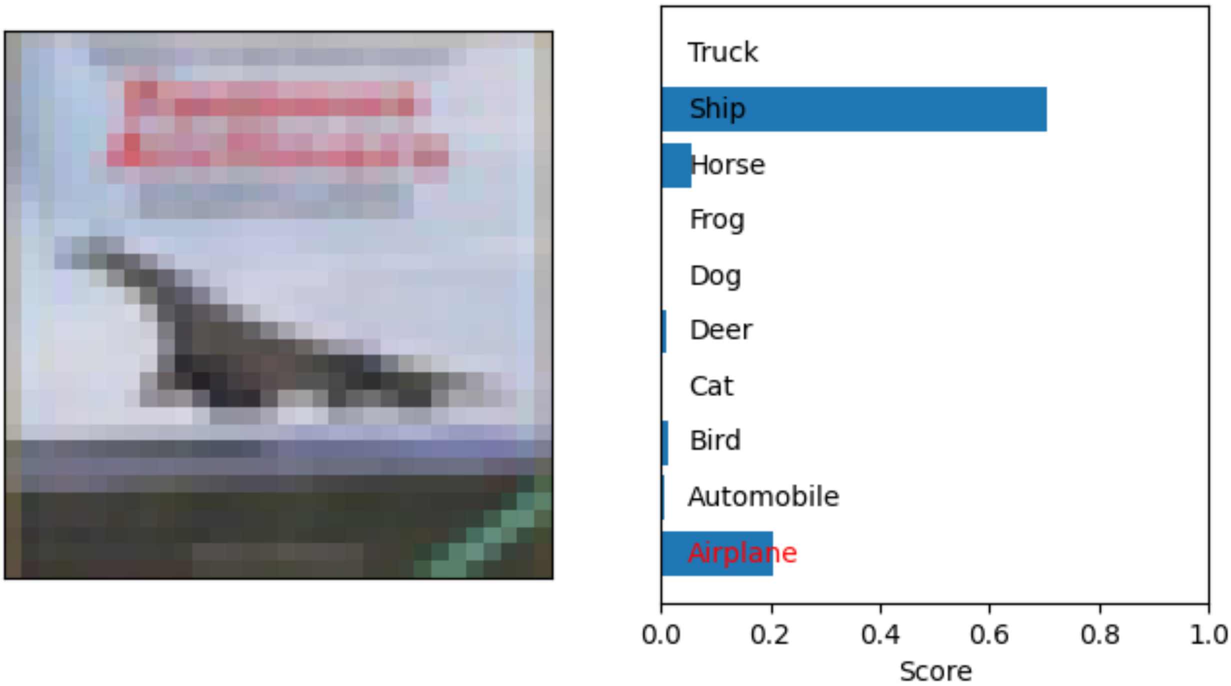


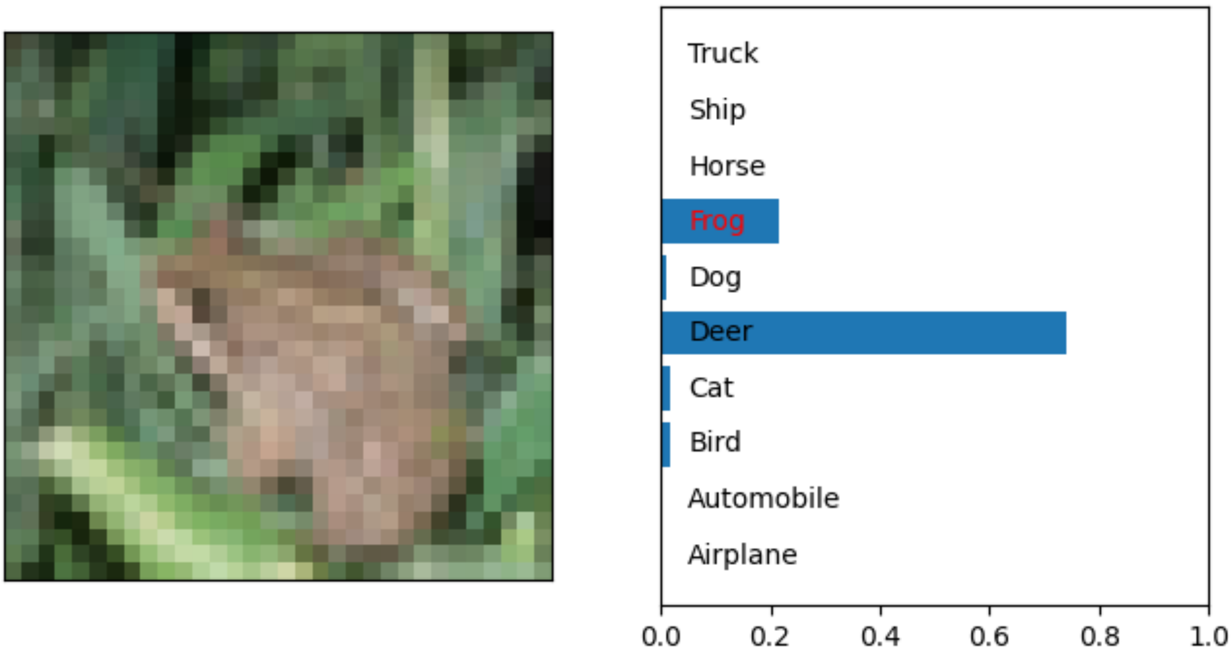
Answer:

Training accuracy: 75%, validation accuracy: 54%, test accuracy: 53%.

```
# Task: plot a few examples of correctly and incorrectly classified images.
# Hint: First find the indices of correctly and incorrectly classified images:
m = y_predict_cl == y_test_cl
i0 = np.arange(8000)[~m]  # misclassified images
i1 = np.arange(8000)[m]  # correctly classified images

# original (unnormalized) test images
x_test = x_test[:8000]

# Hint: Now you can use the `plot_prediction` function to plot the images:
# plot first 3 false classifications
for i in i0[0:3]:
    plot_prediction(x_test[i], y_test_onehot[i], y_predict[i])
```

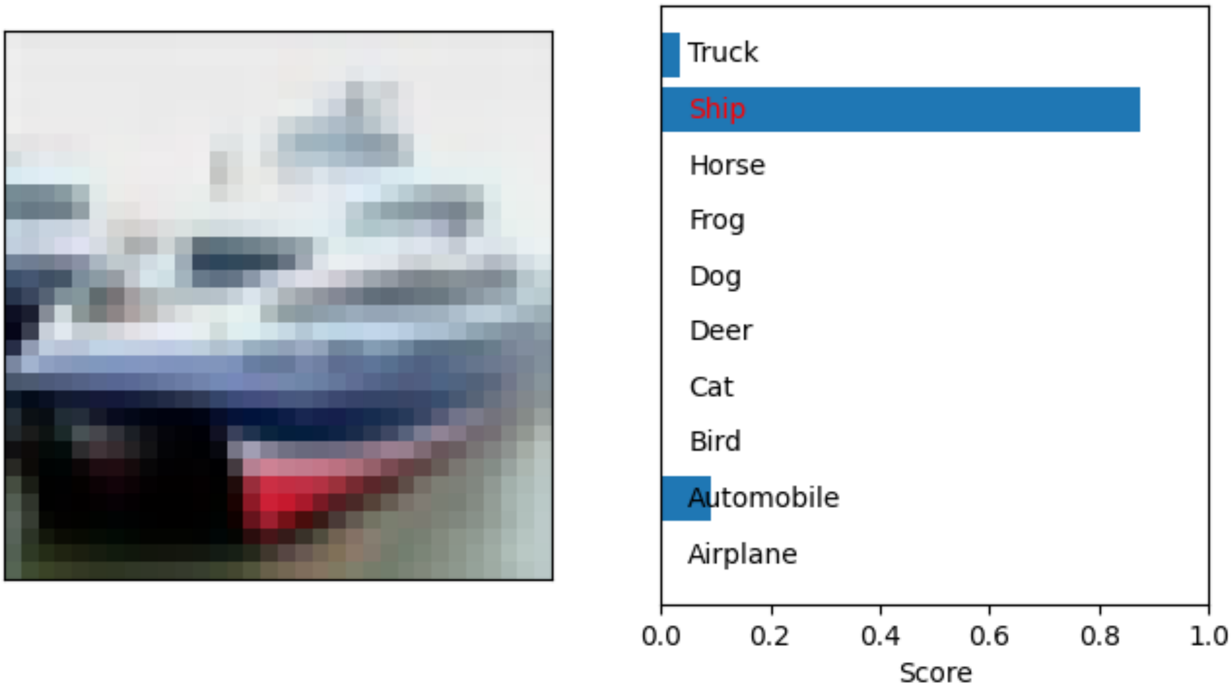

Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
0.0 0.2 0.4 0.6 0.8 1.0
Score

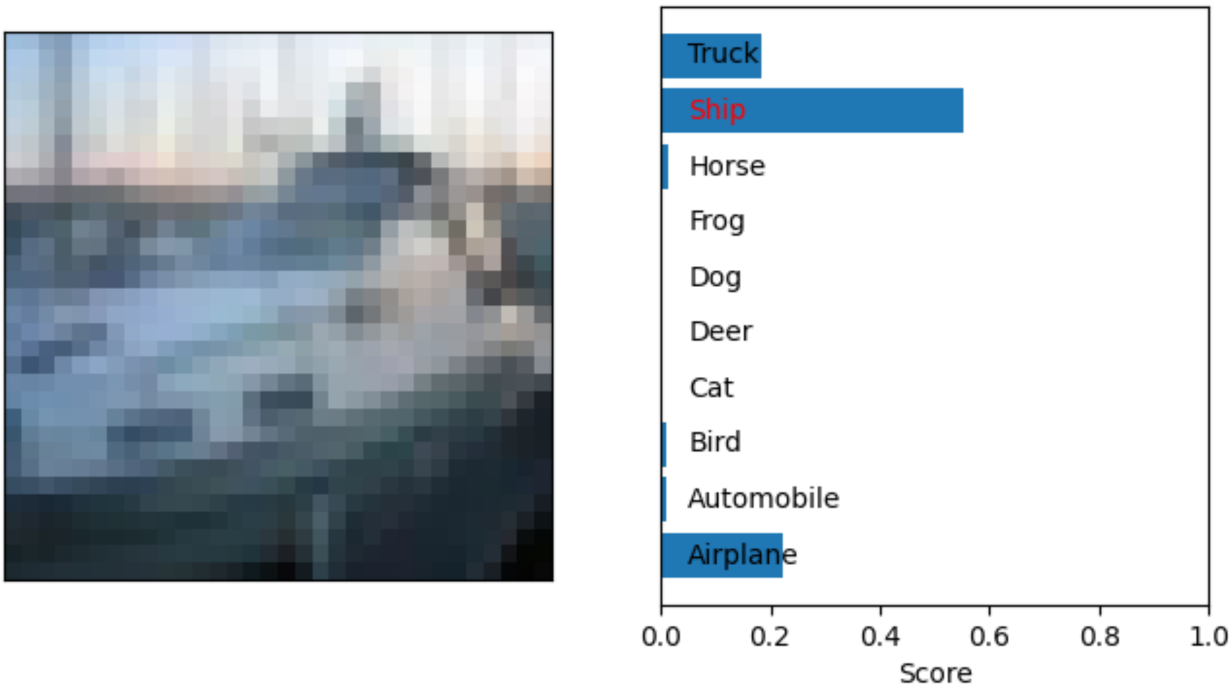

Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
0.0 0.2 0.4 0.6 0.8 1.0
Score

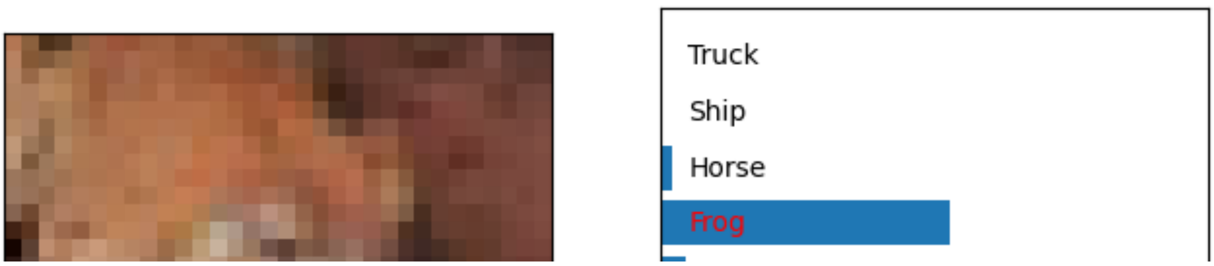

Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
0.0 0.2 0.4 0.6 0.8 1.0

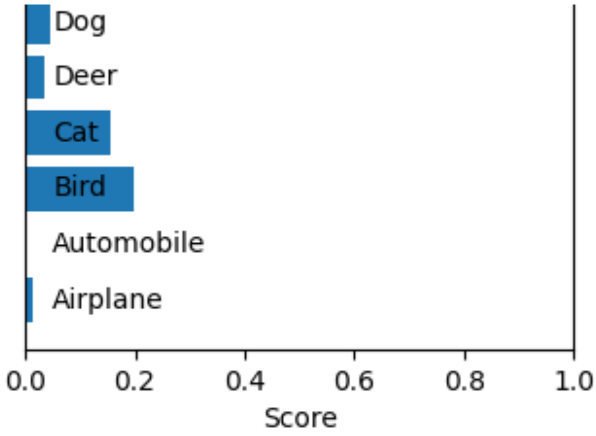
Score

```
#Plot correct classifications:
for i in i1[0:3]:
    plot_prediction(x_test[i], y_test_onehot[i], y_predict[i])
```









CNN In the second part of this exercise, classify the images with a CNN.

```
# Hint: this code snipped shows how to define convolution and maxpooling layers. For m
# https://keras.io/api/layers/convolution_layers/convolution2d/
# https://keras.io/api/layers/pooling_layers/max_pooling2d/
model = tf.keras.models.Sequential(
    [
        layers.Conv2D(16, kernel_size=(3, 3), padding="valid", activation="relu", inpu
        layers.MaxPooling2D((2,2), strides=(2,2)),
        layers.Conv2D(32, kernel_size=(3, 3), padding="valid", activation="relu", inpu
        layers.MaxPooling2D((2,2), strides=(2,2)),
        layers.Flatten(),
        layers.Dense(10, activation="softmax")
    ],
    name="cnn",
)

print(model.summary())
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.p
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "cnn"

Layer (type)	Output Shape	Par
conv2d (Conv2D)	(None, 30, 30, 16)	
max_pooling2d (MaxPooling2D)	(None, 15, 15, 16)	
conv2d_1 (Conv2D)	(None, 13, 13, 32)	4
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 32)	
flatten_1 (Flatten)	(None, 1152)	
dense_3 (Dense)	(None, 10)	11

Total params: 16,618 (64.91 KB)
Trainable params: 16,618 (64.91 KB)
Non-trainable params: 0 (0.00 B)
None

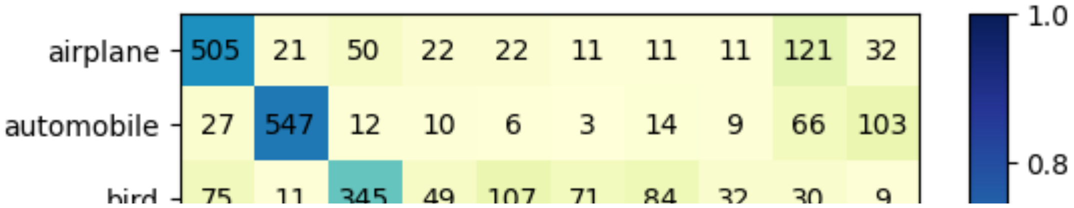
```
# -----------------------------------------------------------
# Training
# -----------------------------------------------------------
model.compile(
    loss='categorical_crossentropy',
    optimizer="adam",
    metrics=["accuracy"]
)

model.fit(
    x_train_norm, y_train_onehot,
    batch_size=2000,
    epochs=20, # train at least for 20 epochs
    verbose=1,
    validation_data=(x_valid_norm, y_valid_onehot),
    callbacks=[tf.keras.callbacks.CSVLogger("history_{}.csv".format(model.name))],
)
```

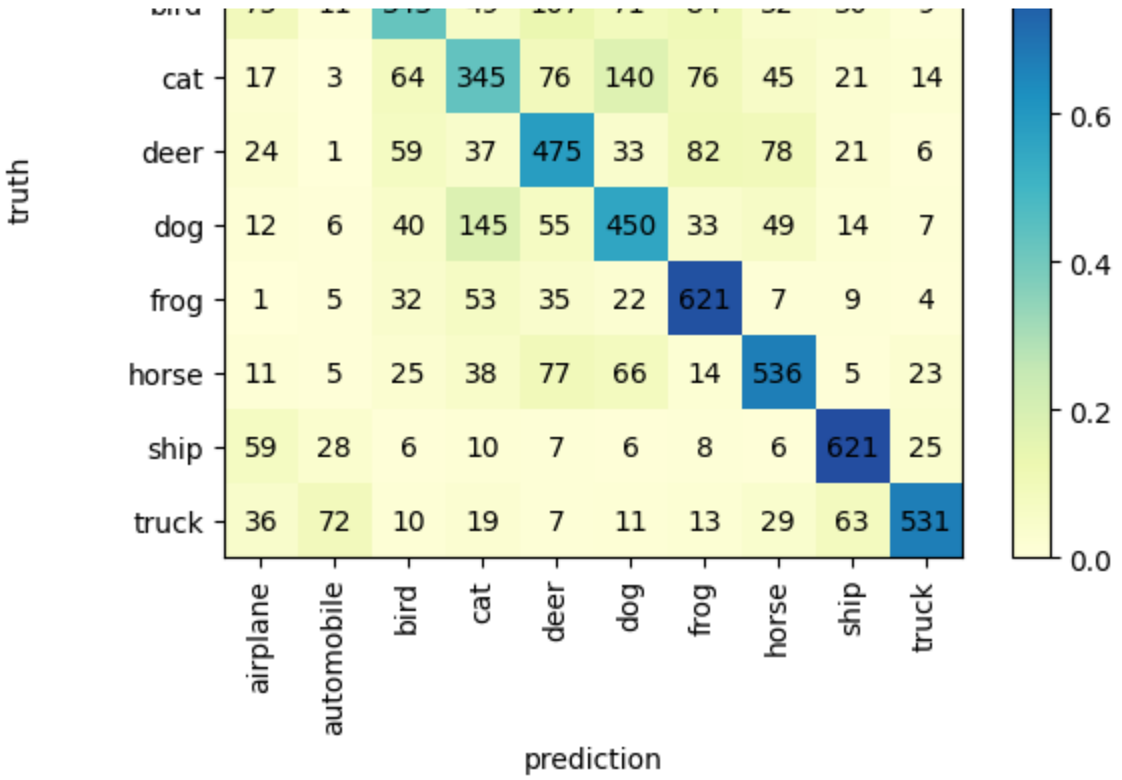
```
Epoch 1/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 34s 1s/step - accuracy: 0.1858 - loss: 2.1984 - val_acc
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.3774 - loss: 1.7506 - val_acc
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.4359 - loss: 1.5899 - val_acc
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.4763 - loss: 1.4875 - val_acc
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.5016 - loss: 1.4155 - val_acc
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.5228 - loss: 1.3629 - val_acc
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.5421 - loss: 1.3176 - val_acc
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.5541 - loss: 1.2751 - val_acc
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.5650 - loss: 1.2464 - val_acc
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.5831 - loss: 1.2049 - val_acc
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 31s 1s/step - accuracy: 0.5855 - loss: 1.1843 - val_acc
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 32s 1s/step - accuracy: 0.5943 - loss: 1.1644 - val_acc
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.6012 - loss: 1.1456 - val_acc
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 35s 1s/step - accuracy: 0.6057 - loss: 1.1370 - val_acc
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.6188 - loss: 1.0997 - val_acc
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.6194 - loss: 1.0952 - val_acc
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.6281 - loss: 1.0789 - val_acc
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.6345 - loss: 1.0647 - val_acc
Epoch 19/20
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 32s 1s/step - accuracy: 0.6335 - loss: 1.0608 - val_acc
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 31s 1s/step - accuracy: 0.6424 - loss: 1.0408 - val_acc
<keras.src.callbacks.history.History at 0x7bab76c48c70>
```

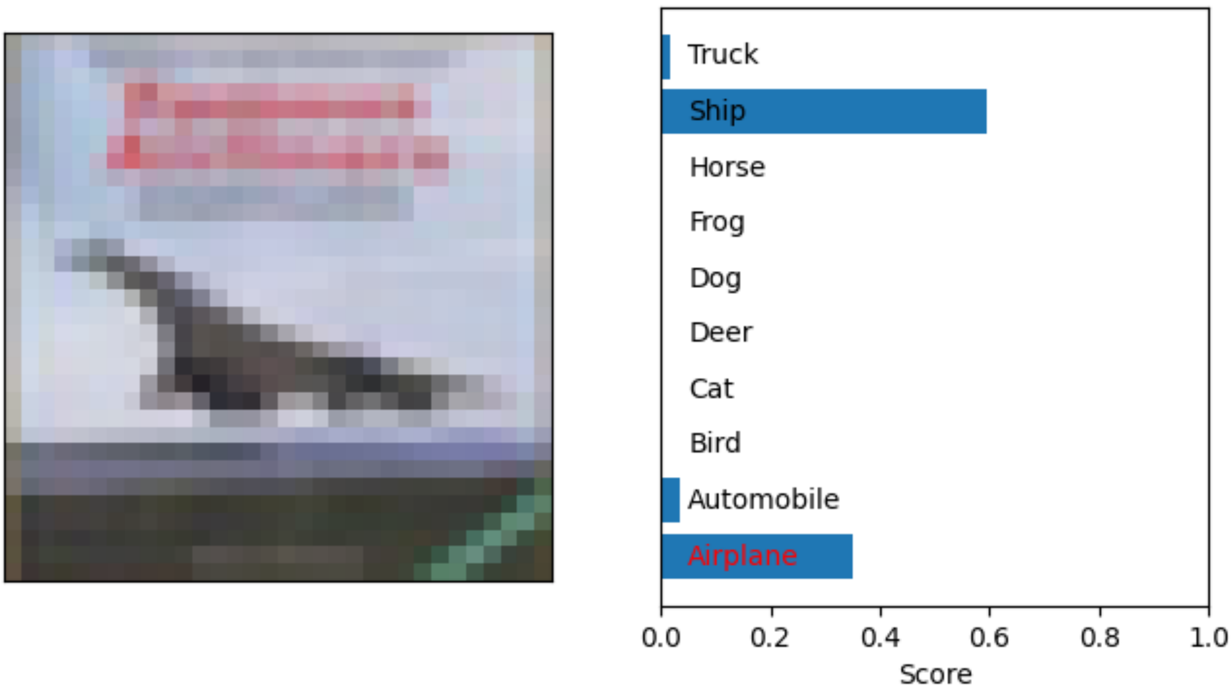
```
# ------------------------------------------------------------
# Plots
# ------------------------------------------------------------
# training curves
history = np.genfromtxt("history_{}.csv".format(model.name), delimiter=",", names=True


# Hint: this is how you can plot the confusion matrix.
# calculate predictions for test set
y_predict = model.predict(x_test_norm, batch_size=128)

# convert back to class labels (0-9)
y_predict_cl = np.argmax(y_predict, axis=1)
y_test_cl = np.argmax(y_test_onehot, axis=1)

# plot confusion matrix
plot_confusion(y_test_cl, y_predict_cl)

#
test_acc=np.mean(np.equal(y_predict_cl, y_test_cl))
print(test_acc)

# Task: plot a few examples of correctly and incorrectly classified images.
# Hint: First find the indices of correctly and incorrectly classified images:
m = y_predict_cl == y_test_cl
i0 = np.arange(8000)[~m]  # misclassified images
i1 = np.arange(8000)[m]  # correctly classified images

# original (unnormalized) test images
x_test = x_test[:8000]

# Hint: Now you can use the `plot_prediction` function to plot the images:
# plot first 3 false classifications
for i in i0[0:3]:
    plot_prediction(x_test[i], y_test_onehot[i], y_predict[i])

#Plot correct classifications:
for i in i1[0:3]:
    plot_prediction(x_test[i], y_test_onehot[i], y_predict[i])
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step
0.622
```

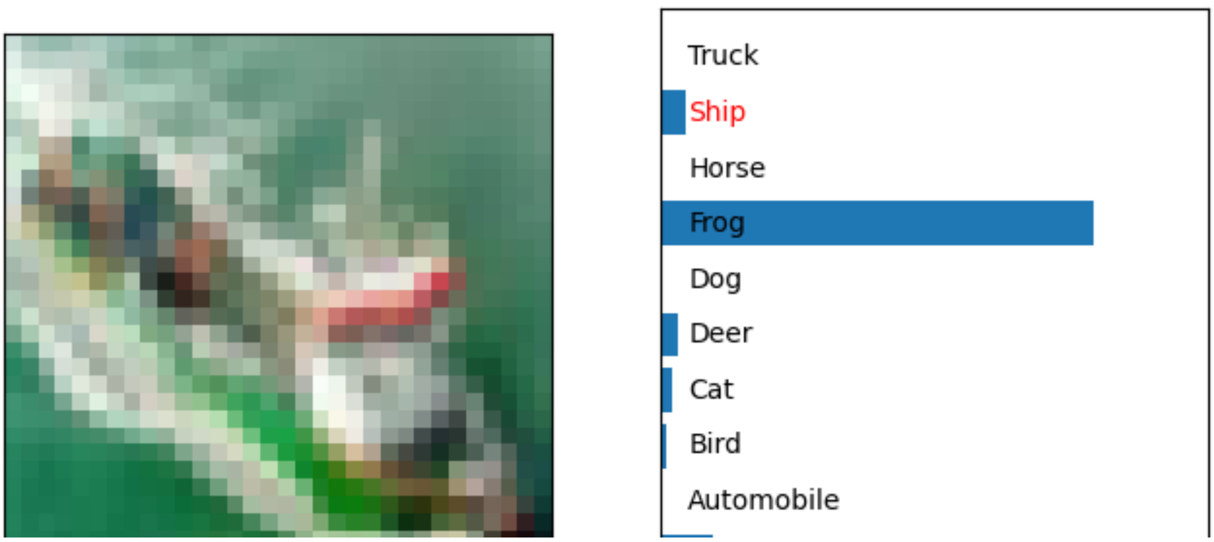




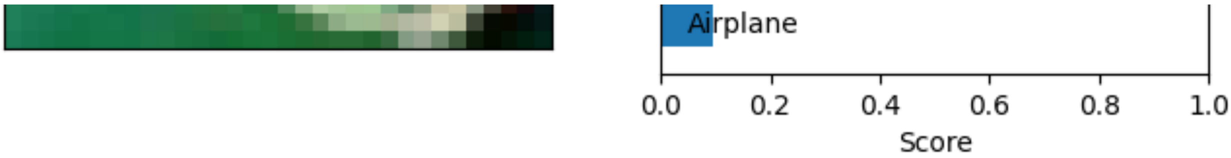
truth
cat 17 3 64 345 76 140 76 45 21 14
deer 24 1 59 37 475 33 82 78 21 6
dog 12 6 40 145 55 450 33 49 14 7
frog 1 5 32 53 35 22 621 7 9 4
horse 11 5 25 38 77 66 14 536 5 23
ship 59 28 6 10 7 6 8 6 621 25
truck 36 72 10 19 7 11 13 29 63 531
airplane automobile bird cat deer dog frog horse ship truck
prediction
0.0 0.2 0.4 0.6

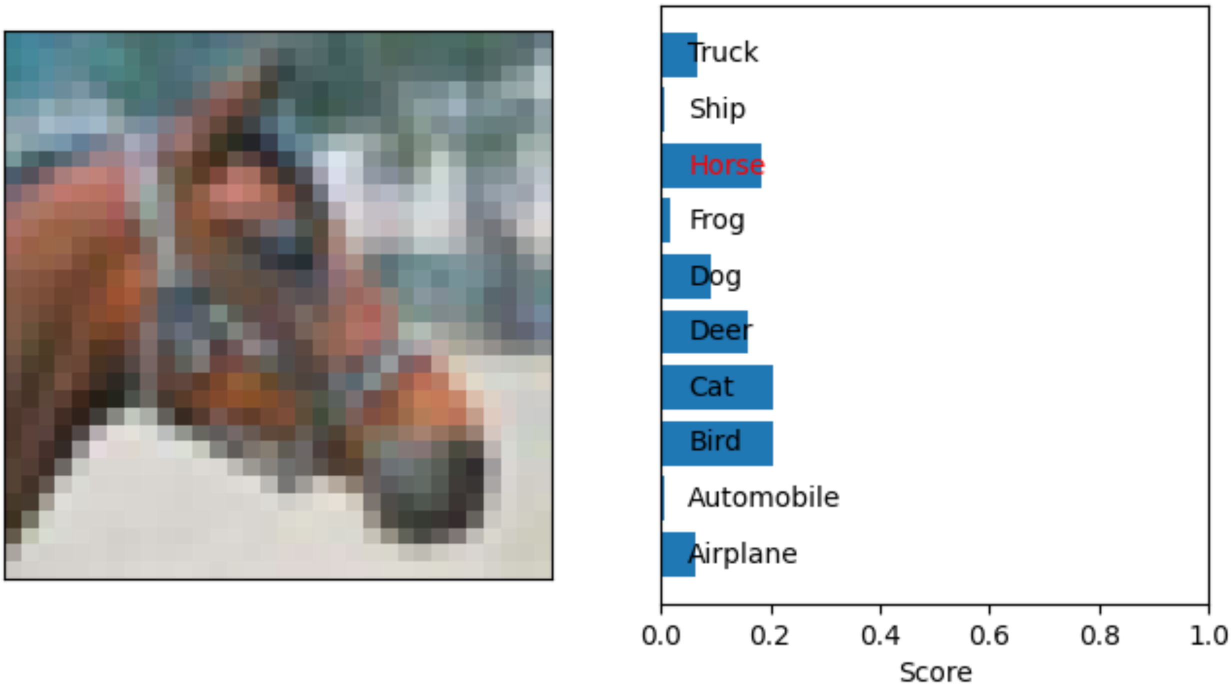


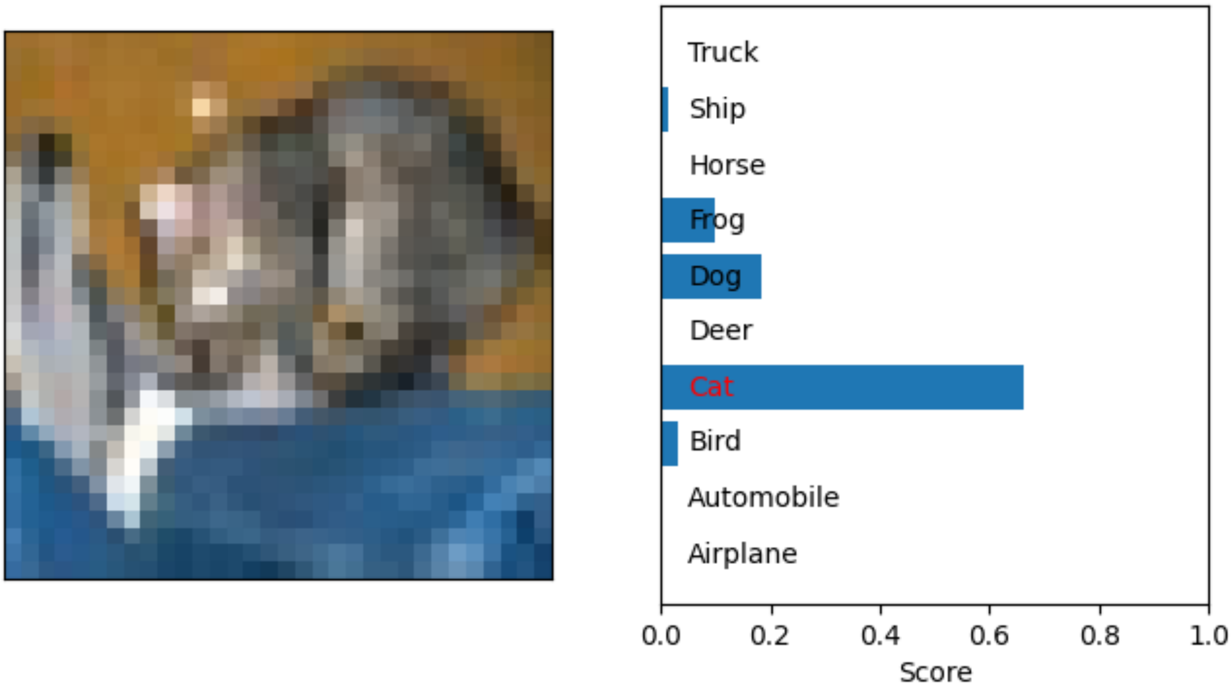
Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
0.0 0.2 0.4 0.6 0.8 1.0
Score

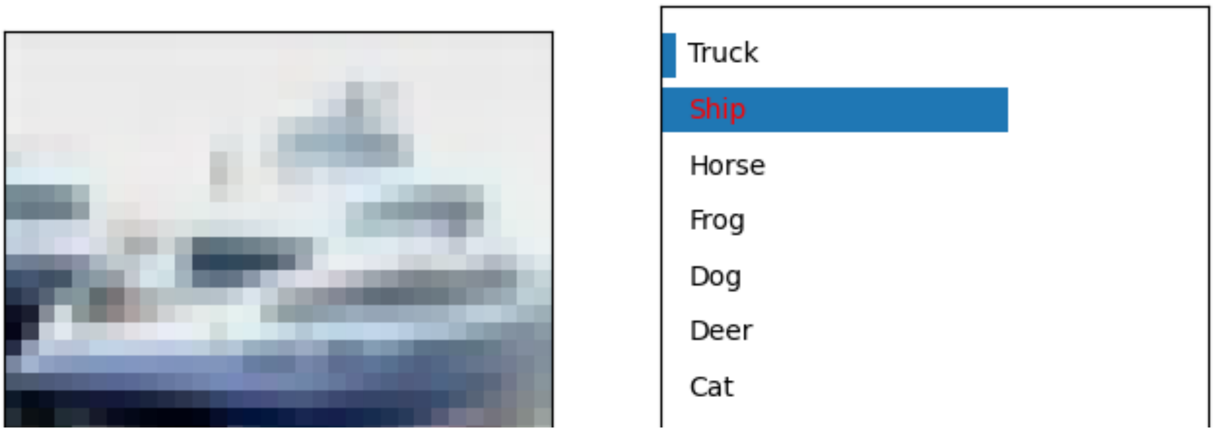


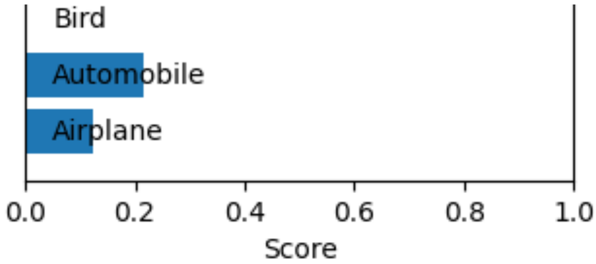
Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile

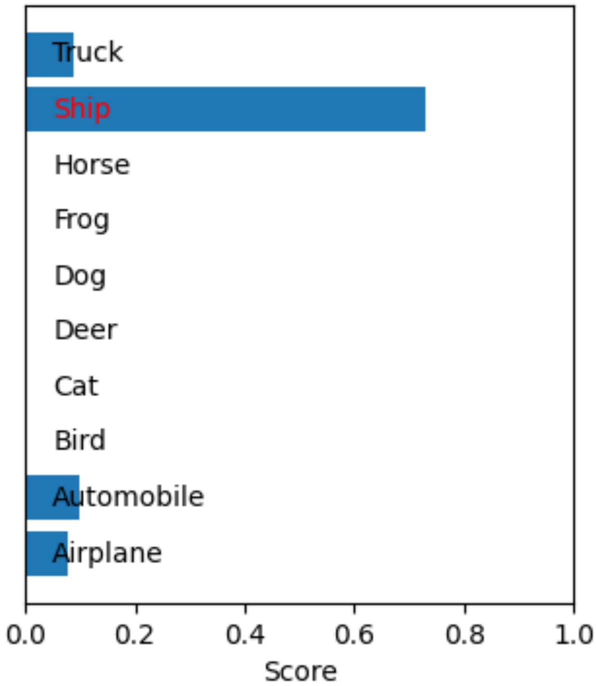

Airplane
0.0
0.2
0.4
0.6
0.8
1.0
Score


Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
0.0
0.2
0.4
0.6
0.8
1.0
Score


Truck
Ship
Horse
Frog
Dog
Deer
Cat
Bird
Automobile
Airplane
0.0
0.2
0.4
0.6
0.8
1.0
Score


Truck
Ship
Horse
Frog
Dog
Deer
Cat





Note

train accuracy: 64%, validation accuracy: 59%, test accuracy: 62%. Much better test accuracy now=better model with cnn.