TensorFlow Cheat Sheet: Beginner to Advanced

1. Library Structure Overview

```
tensorflow/
├── keras                # High-level APIs (models, layers, training)
│   ├── models           # Sequential, functional API
│   ├── layers           # Dense, Conv2D, LSTM, etc.
│   ├── losses           # Loss functions (categorical, mse, etc.)
│   ├── optimizers       # Adam, SGD, RMSprop
│   ├── metrics          # Accuracy, Precision, Recall, etc.
│   └── callbacks        # EarlyStopping, ReduceLROnPlateau, etc.
├── data                 # tf.data: Efficient input pipelines
├── lite                 # TFLite: Mobile & embedded deployment
├── saved_model          # Model saving/loading (serialization)
├── tf.function          # Graph-based computation (autograph)
├── distribute           # Distributed/multi-GPU/TPU training
└── others: text, image, Estimator, Hub, Addons...
```

2. Getting Started: Setup & Data

```
import tensorflow as tf
import numpy as np
print(tf.__version__)
```

Data Handling

```
x = np.random.rand(100, 3)
y = (x.sum(axis=1) > 1.5).astype(int)
ds = tf.data.Dataset.from_tensor_slices((x, y)).batch(16).shuffle(100)
```

Load from CSV/TFRecord, etc.

```
tfrecord_ds = tf.data.TFRecordDataset(['file.tfrecord'])
```

3. Building Models (Keras API)

a. Sequential Model

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Dense(64, activation='relu', input_shape=(3,)),
    layers.Dense(1, activation='sigmoid')
])
```

b. Functional API (Complex, multi-input/output)

```
inputs = layers.Input(shape=(10,))
h = layers.Dense(32, activation="relu")(inputs)
outputs = layers.Dense(1)(h)
model = models.Model(inputs, outputs)
```

c. Custom Model (Subclassing)

```
class MyModel(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.dense1 = layers.Dense(32, activation="relu")
        self.dense2 = layers.Dense(1)
    def call(self, x):
        return self.dense2(self.dense1(x))
model = MyModel()
```

4. Compiling & Training Models

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
model.fit(x, y, epochs=10, batch_size=16, validation_split=0.2)
```

With tf.data pipeline:

```
model.fit(ds, epochs=10)
```

Callbacks

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=3),
    tf.keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True)
]
model.fit(x, y, callbacks=callbacks)
```

5. Evaluate, Predict, and Save/Load

```
model.evaluate(x_test, y_test)
preds = model.predict(x_new)
model.save('model_folder')              # Save full model (SavedModel format)
restored = tf.keras.models.load_model('model_folder')
```

6. Core API: Tensors, Ops, Autograph

```
a = tf.constant([[1,2],[3,4]])
b = tf.Variable(tf.ones((2,2)))
c = tf.add(a, b)
with tf.GradientTape() as tape:
    y = tf.math.square(a)
    grad = tape.gradient(y, a)
```

- Use `tf.function` decorator for high-performance “compiled” functions:

```
@tf.function
def f(x):
    return x * x + 5
```

7. Layers and Custom Components

Common Layers

- `Dense`, `Conv2D`, `LSTM`, `Dropout`, `BatchNormalization`, etc.

Custom Loss or Metric

```
def custom_loss(y_true, y_pred):
    return tf.reduce_mean(tf.square(y_true - y_pred))
model.compile(loss=custom_loss, ...)
```

Custom Layer

```
class MyLayer(tf.keras.layers.Layer):
    def call(self, inputs):
        return tf.nn.relu(inputs)
```

8. Preprocessing & Data Pipelines

- `tf.keras.layers.Normalization` for scaling.
- `tf.keras.preprocessing.image.ImageDataGenerator` for augmentation.
- `tf.data.Dataset` for efficient, parallelized, multi-format input loading and transformation.

9. Model Inspection & Explainability

- Model summary: `model.summary()`
- Visualize architecture: `tf.keras.utils.plot_model(model)`
- Feature importance: Use SHAP, tf-explain, or interpret feature attributions post-training.

10. Advanced Topics

a. Distributed and Multi-GPU Training

```
strategy = tf.distribute.MirroredStrategy()
with strategy.scope():
    model = build_model_fn()
    model.compile(...)
    model.fit(...)
```

b. Mixed Precision

```
tf.keras.mixed_precision.set_global_policy('mixed_float16')
```

c. Serving and Export

- Export for production: `model.save(...)`
- Serve via TensorFlow Serving, export as `SavedModel`
- For edge/mobile: `tf.lite.TFLiteConverter.from_saved_model('folder')` → Deploy TFLite

d. TensorBoard (Visualization, Debugging)

```
tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir='./logs')
model.fit(x, y, callbacks=[tensorboard_cb])
# Run: tensorboard --logdir=./logs
```

e. Transfer Learning

```
base = tf.keras.applications.ResNet50(weights='imagenet', include_top=False,
input_shape=(224,224,3))
for layer in base.layers:
    layer.trainable = False
x = layers.GlobalAveragePooling2D()(base.output)
output = layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs=base.input, outputs=output)
```

f. NLP & Vision Advanced

- **Text:** `tf.keras.layers.Embedding`, `tf.data.TextLineDataset`, `tf.keras.layers.MultiHeadAttention`
- **Vision:** `tf.keras.layers.Conv2D`, `tf.image`, data augmentation layers

g. Low-Level Operations

```
# Eager mode: immediate execution (default since TF 2.0)
# Graph mode: wrap code with @tf.function for graph optimizations
```

11. Integration and Ecosystem

- **tf.keras:** High-level modeling API (recommended for most users)
- **tf.data:** Data pipelines for batching, shuffling, mapping, and prefetching

- **tf.lite:** Model conversion & inference for mobile/edge
- **tf.distribute:** Multi-device/machine training
- **tf.summary:** Logging for TensorBoard

12. Useful References

- Official guides/tutorials: tensorflow.org
- tf.keras API Reference
- Community tutorials (TensorFlow Hub, Coursera)
- Example notebooks (TensorFlow Model Garden)

Example: End-to-End Binary Classification

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=16)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Build
model = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(16,)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile & Train
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=8, batch_size=32, validation_split=0.2)

# Evaluate & Save
print(model.evaluate(X_test, y_test))
model.save('demo_model')
```